



\$9.95 U.S.
Display Until
7/15/93

os/2 *Developer*

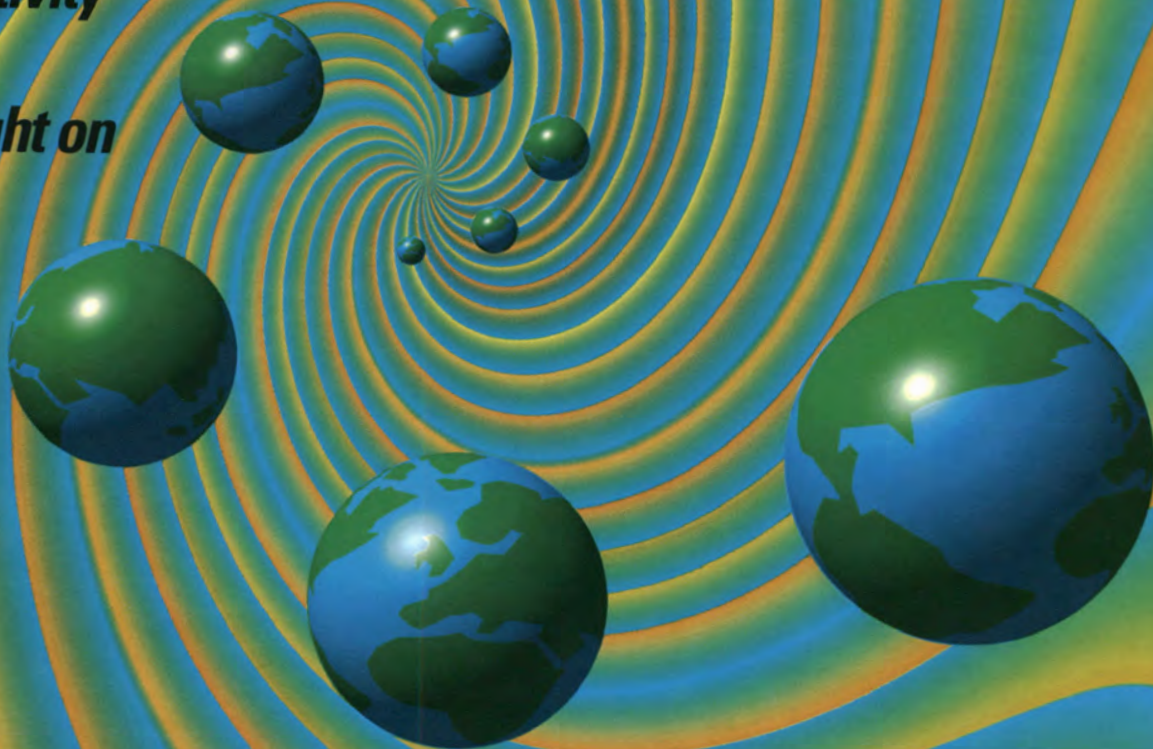
THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

Vol. 5 No. 2

 **Communications
And
Connectivity**

 **Spotlight on
Oracle
Corp.**

 **More
On
SOM**



Delivering the Power: WATCOM C9.0/386

► The Widest Range of 32-bit Intel x86 Platforms

32-bit DOS, 32-bit Windows, OS/2 2.0, AutoCAD ADS

► The Industry's Leading Code Optimizer

Advanced global optimizer with new 486 optimizations

► The Most Comprehensive Toolset

Debugger, profiler, protected-mode compiler and linker, 32-bit DOS extender with royalty-free run-time, licensed components from Microsoft SDK, and more

► The Best Value in 32-Bit Tools: \$895*

Unleash 32-bit Power!

WATCOM C9.0/386 lets you exploit the two key 32-bit performance benefits. The 32-bit flat memory model simplifies memory management and lets applications address beyond the 640K limit. Powerful 32-bit instruction processing delivers a significant speed advantage: typically at least a 2x speedup.

You Get:

- **100% ANSI and SAA compatible:** C9.0/386 passes all Plum Hall Validation Suite tests
- **Extensive Microsoft compatibility** simplifies porting of 16-bit code
- **Royalty-free** run-time for 32-bit DOS, Windows and OS/2 apps
- **Comprehensive toolset** includes debugger, linker, profiler and more
- **DOS extender** support for Rational, Phar Lap and Ergo
- **Run-time compatible with WATCOM FORTRAN 77/386**

32-bit DOS support includes the DOS/4GW 32-bit DOS extender by Rational Systems with royalty-free runtime license

- Virtual Memory support up to 32Mb

32-bit Windows support enables development and debugging of true 32-bit GUI applications and DLLs.

- Includes licensed Microsoft SDK components

32-bit OS/2 2.0 support includes development for multiple target environments including OS/2 2.0, 32-bit DOS and 32-bit Windows

- Access to full OS/2 2.0 API including Presentation Manager
- Integrated with IBM Workframe/2 Environment

AutoCAD ADS and **ADI** Development: Everything you need to develop and debug ADS and ADI applications for AutoCAD Release 11

Novell's *Network C for NLN's SDK* includes C/386

The Industry's Choice.

Autodesk, *Robert Wenig, Manager, AutoCAD for Windows:*

"At Autodesk, we're using WATCOM C/386 in the development of strategic new products since it gives us a competitive edge through early access to new technologies. We also highly recommend WATCOM C/386 to third party AutoCAD add-on (ADS and ADI) developers."

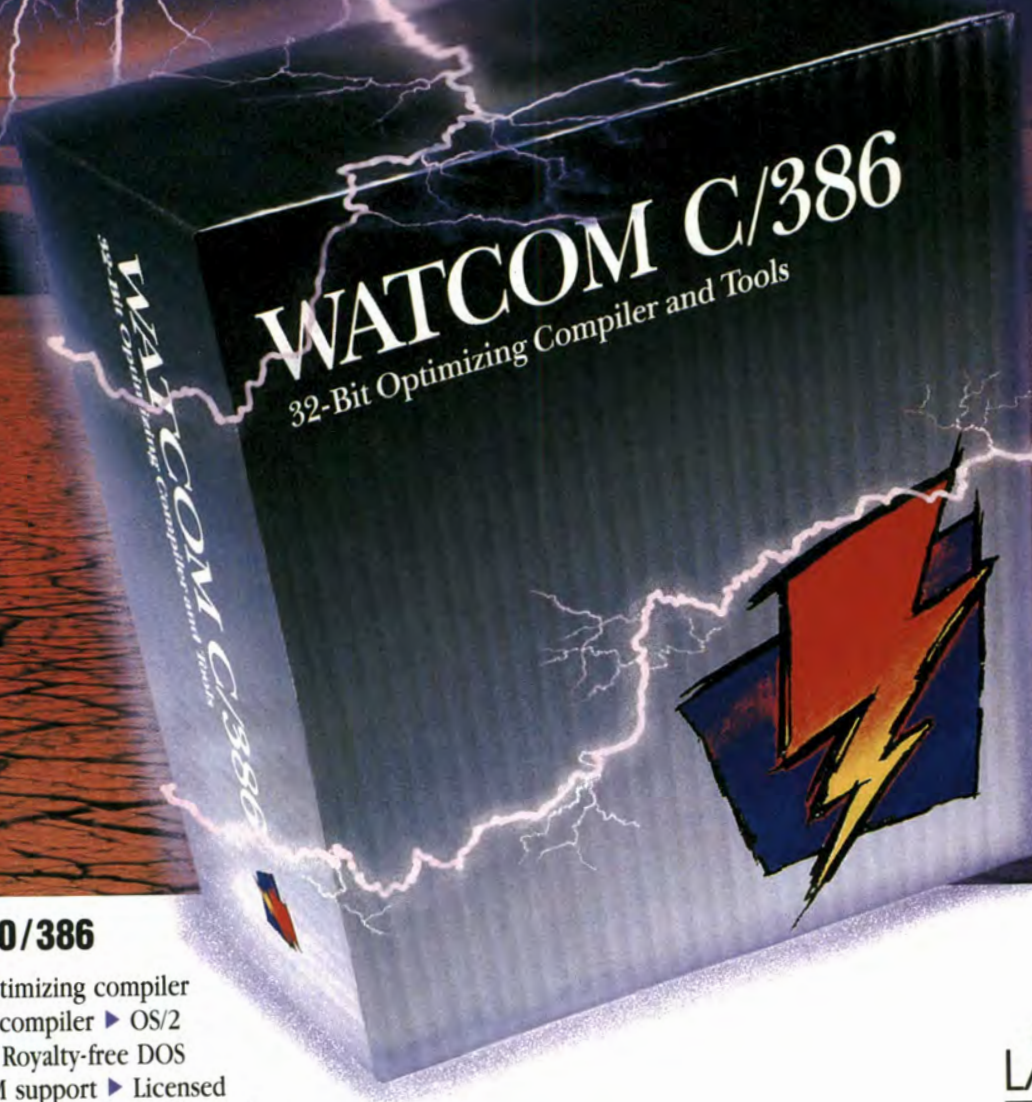
Fox Software, *David Fulton, President:* "FoxPro 2.0 itself is written in WATCOM C, and takes advantage of its many superior features. Optimizing for either speed or compactness is not uncommon, but to accomplish both was quite remarkable."

GO, *Robert Carr, Vice President of Software:* "After looking at the 32-bit Intel 80x86 tools available in the industry, WATCOM C was the best choice. Key factors in our decision were performance, functionality, reliability and technical support."

IBM, *John Soyering, Director of OS/2 Software Developer Programs:* "IBM and WATCOM are working together closely to integrate these compilers with the OS/2 2.0 Programmer's Workbench."

Lotus, *David Reed, Chief Scientist and Vice President, Pen-Based Applications:* "In new product development we're working with WATCOM C because of superior code optimization, responsive support, and timely delivery of technologies important to us like p-code and support for GO Corp's. PenPoint."

Novell, *Nancy Woodward, V.P. and G.M., Development Products:* "We searched the industry for the best 386 C compiler technology to incorporate with our developer toolkits. Our choice was WATCOM."



WATCOM C9.0/386

- ▶ 100% ANSI C optimizing compiler
- ▶ Protected-mode compiler ▶ OS/2 hosted-compiler ▶ Royalty-free DOS extender with VMM support ▶ Licensed components of the Microsoft Windows SDK
- ▶ Interactive source-level debugger ▶ Linker
- ▶ Protected-mode linker ▶ OS/2-hosted linker ▶ Profiler
- ▶ Object code librarian ▶ Object code disassembler ▶ MAKE facility ▶ Patch facility ▶ Object module convert utility
- ▶ Windows supervisor ▶ Bind facility for Windows applications
- ▶ 32-bit run-time library object code ▶ Special 32-bit libraries for Windows API ▶ Graphics library for Extended DOS applications ▶ 32-bit Run-time libraries for Windows ▶ 32-bit Run-time libraries for OS/2

Special Offer

Buy **WATCOM C9.0/386** and you'll be eligible to obtain:

WATCOM C9.0 Delta Pack provides you with the tools necessary to develop and debug 16-bit applications for DOS, Windows, and OS/2. **Only \$99.** (\$495 comparative separate cost)

WATCOM FORTRAN 77/386 Delta Pack provides you with everything necessary to develop and debug 32-bit FORTRAN applications for extended DOS, 32-bit Windows and OS/2 2.0. **Only \$399.** (\$895 comparative separate cost)



WATCOM

1-800-265-4555

The Leader in 32-bit Development Tools

415 Phillip Street, Waterloo, Ontario, Canada
 Telephone: (519) 886-3700, Fax: (519) 747-4971
 *Price does not include freight and taxes where applicable. Authorized dealers may sell for less.
 WATCOM C and Lightning Device are trademarks of WATCOM Systems Inc.
 DOS/4G and DOS/16M are trademarks of Rational Systems Inc.
 Other trademarks are the properties of their respective owners.
 Copyright 1992 WATCOM Products Inc.

Bring GUIs to your COBOL applications and keep your interface options open with Micro Focus Dialog System™ 2.1.



Dialog System makes it possible to prototype, develop and customize graphical user interfaces without learning complicated Application Programming Interfaces (APIs). Programmers familiar with COBOL can be up and running quickly with Dialog System.

And which "industry-standard" user interface will best suit the end users at your site? Will it be Microsoft® Windows™? OS/2® Presentation Manager™? OSF/Motif™? Or perhaps character-mode interfaces for DOS and UNIX®?

When you develop user interfaces with Dialog System it doesn't matter, because a single user interface can be portable across all these environments.

Dialog System supports "point and click" development of graphical and character-based user interfaces. It isolates screen and keyboard logic from



COBOL applications, replacing hundreds of lines of screen definition code with a simple CALL statement. That means smaller, faster and more reusable COBOL programs.

Dialog System user interfaces can be prototyped, developed and updated without impacting the application logic. When a prototype interface is completed, it can be put directly into production without any additional development.

Bring the excitement of GUIs to your COBOL applications with Micro Focus COBOL and Dialog System.

Call Micro Focus at 800-872-6265 and learn how you can bring GUIs to your COBOL applications. Discover "A Better Way of Programming™" with Micro Focus.

MICRO FOCUS

Micro Focus Inc., 2465 East Bayshore Road, Palo Alto, CA 94303. Tel. (415) 856 4161.

Micro Focus is a registered trademark of Micro Focus. Dialog System and "A Better Way of Programming" are trademarks of Micro Focus. All other trademarks are property of their respective companies.

GSA Contract Number GS00K90AGS5251-PS02.

IBM **OS/2** Developer

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

SPRING 1993



EDITOR'S COMMENTS

New Responsiveness And A New Look **5**



SPOTLIGHT

Oracle: OS/2 Preferences And Portability **6**



COMMUNICATIONS

Automating Application Installability With Communications Manager/2 **17**

Extending Application Interfaces For Communications: Client/Server **27**

Application Design For Remote Access **39**



GUI

Programming the OS/2 Container Control: By Example **48**

Designing Custom Controls **72**



OBJECT-ORIENTED PROGRAMMING

Workplace Shell Programming Using Multiple Processes **60**

A Bird's Eye View of SOM **86**

Method Resolution In SOM **94**

Using Name-Lookup Resolution In SOM **103**



LAN

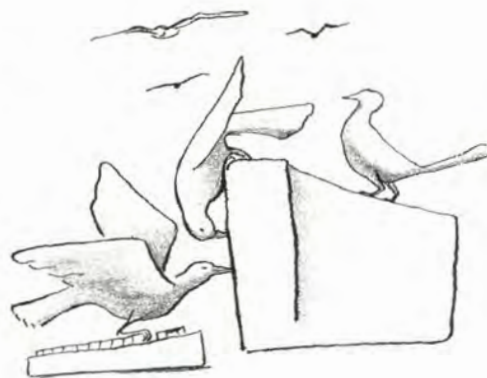
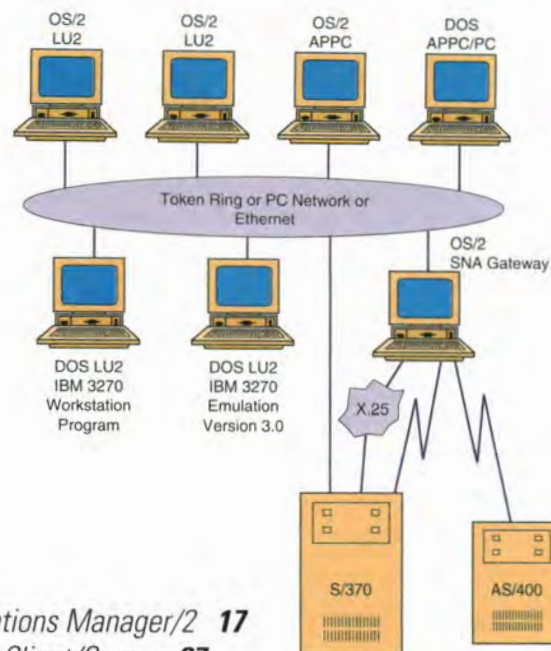
Network Plotting Formation: The LAN NetView Start Objects **113**

Server-Based Systems: Protection From Disk Failures **123**



32-BIT

Writing Memory Allocation Functions With DosSubAllocMem **127**



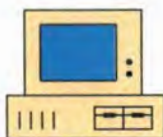
ORACLE Server
for OS/2



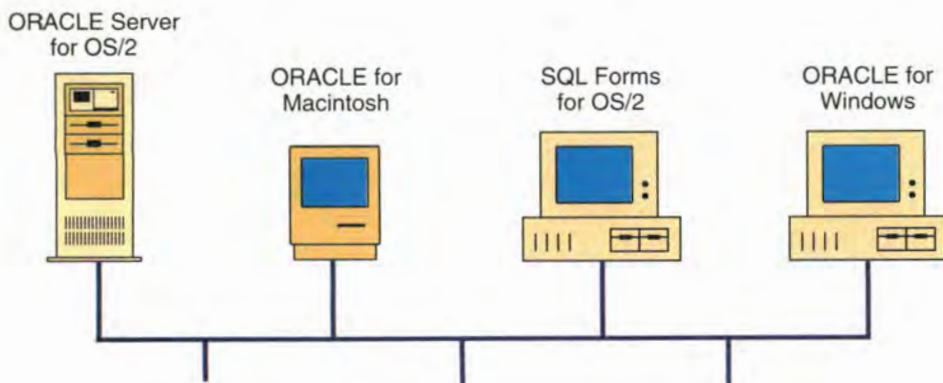
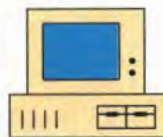
ORACLE for
Macintosh



SQL Forms
for OS/2



ORACLE for
Windows



EDITOR

Dick Conklin

EXECUTIVE EDITOR

Nicole Freeman

PRODUCTION EDITOR

Peter Altenberg

COPY/PRODUCTION EDITOR

Lisa Gluskin

GROUP DIRECTOR

Don Pazour

ASSOCIATE PUBLISHER

Cathy Passage

ADVERTISING SALES

Veronica Costanza

(212) 626-2418

Jo Ben-Atar

(203) 498-0329

Michele Blake

(212) 626-2322

Dave Moreau

(212) 626-2318

MARKETING MANAGER

Susan McDonald

ART DIRECTOR/MARKETING

Christopher H. Clarke

ADVERTISING PRODUCTION

COORDINATOR

Tom Marzella

DIRECTOR OF PRODUCTION

Andy Mickus

DIRECTOR OF CIRCULATION

Jerry Okabe

CIRCULATION DIRECTOR

Moiria Boyle

SUBSCRIPTION INQUIRIES

U.S.: (800) WANT-OS2

Foreign: (708) 647-5960

PRINTED IN THE USA

IBM OS/2 Developer is published by Miller Freeman Inc., 600 Harrison Street, San Francisco, CA 94107, (415) 905-2200, Dick Conklin, Editor.

The terms "OS/2" and "OS/2 2.0" are used in this publication as abbreviations for the full term "OS/2 operating system." OS/2 is a registered trademark of the IBM Corporation.

Titles and abstracts, but no other portions, of information in this publication may be copied and distributed by computer-based and other information service systems. Permission to republish information from this publication in any other publication of computer-based information systems must be obtained from the Editor.

IBM believes the statements contained herein are accurate as of the date of publication of this document. However, IBM, hereby disclaims all warranted either expressed or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings or other incidental or consequential damage arising out of the use or inability to use any information provided through this publication even if IBM has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages so the above limitation or exclusion may not apply to you.

This publication may contain technical inaccuracies or typographical errors. Also, illustrations contained here may show prototype equipment. Your system configuration may differ slightly.

Some publications referenced in these articles have an IBM order number. They may be ordered by calling (800)879-2755 in the U.S. and selecting option #1. Payment can be made with a major credit card. IBM employees should order publications on PUBORDER.

IBM, OS/2, C Set/2, Presentation Manager, Workplace Shell, Communications Manager/2, Distribution Manager/2, Enterprise Systems Architecture/390, Application System 400, RISC/6000, Distributed Console Access Facility, NetView, Extended Services, LAN Server, Advanced Network Transport Services/2, and DASD Manager are registered trademarks of IBM Corp. The IBM logo is a registered trademark of IBM Corp.

CompuServe is a registered trademark of CompuServe Inc.

This publication may contain articles by non-IBM authors. These articles represent the views of their authors. IBM does not endorse any non-IBM products that may be mentioned. Questions should be directed to the authors.

This information is not intended to be an assertion of future action. IBM expressly reserves the right to change or withdraw current products that may or may not have the same characteristics or codes listed in this publication. Should IBM modify its products in a way that may affect the information contained in any user of the modification. It is possible, that this material may contain reference to, or information about, IBM products (machines and programs), programming or services that are not be construed to mean that IBM intends to announce such products, programming or services in your country.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. All specifications are subjects to change without notice.

With respect to advertising material herein, this publication does not constitute an expressed or implied recommendation or endorsement by IBM of any particular product, service, company, or technology.

IBM takes no responsibility whatsoever with regard to the selection, performance or use of the products advertised herein. All understanding, agreements, or warranties must take place directly between the software vendors and prospective users.

To correspond with the IBM OS/2 Developer, please write to the Editor at IBM Corp., Internal Zip 2230, P.O. Box 1328, Boca Raton, FL 33429-1328.

Internet is a registered trademark of Internet, Inc.

Windows and Microsoft C are registered trademarks of Microsoft Corp.

ORACLE Server, ORACLE7, CASE*Designer, CASE*Dictionary, SQL*Net, SQL*Net SPX, SQL NetBIOS, SQL*Net TCP/IP, SQL*NetDECNet, SQL*Net APPC, and KEDIT are trademarks of Oracle Corp.

Smalltalk/V PM is a trademark of Digitalk, Inc.

IBM OS/2 Developer: Vol. 5, No. 2, Spring 1993

Subscription information: U.S.: One year (four issues), \$39.95. Canada, Mexico, and international surface mail, add \$16 per year for postage. Canadian GST no. 124513185. International air mail, add \$30 per year for postage. Foreign orders must be accompanied by payments in U.S. funds. For new orders and customer service, call (800) WANT-OS2 (800-926-8672) or (708) 647-5960 (fax: 708-647-0537). IBM employees and branch office customers can subscribe to the IBM OS/2 Developer through IBM Mechanicsburg's Systems Library Services (SLSS) using the IBM OS/2 Developer's order number: G362-0001. POSTMASTER: Please send address changes to IBM OS/2 Developer, P.O. Box 1079, Skokie, IL 60076-8079. Allow eight weeks for subscriptions to begin. The IBM OS/2 Developer is published quarterly by Miller Freeman Inc., 600 Harrison Street, San Francisco, CA 94107, (415) 905-2200. Application to mail at second class postage rates is pending at San Francisco, CA and additional mailing offices.

© Copyright 1993 by Miller Freeman Inc. Printed in U.S.A.



BY DICK CONKLIN

Editor's Comments

New Responsiveness And A New Look

AS YOU CAN SEE, *OS/2 DEVELOPER*'S LOOK IS changing; we introduce a new page format with this issue, one that allows us to fit more material in an issue. This facelift is in line with our goal to deliver more useful information about OS/2™ application development.

WE HEAR YOU!

In a recent survey of members of our Developer Assistance Program, *OS/2 Developer* received a 98% approval rating—thanks! Several of you commented on our editorial content, asking for more “tips” and “in-depth technical articles.” Those requests are consistent with what you’ve been telling us on CompuServe™, and our authors have been responding to your suggestions with some “meaty” articles.

Our authors have also been uploading their sample source code to CompuServe's OS2DF2 library and the IBM NSC BBS. Uploaded files help reduce the long code listings in the magazine and make it easier to get at the code without entering it yourself. A good example is the package from the “Demystifying Custom Controls” article by Mark Bengé and Matt Smith in our Winter '93 issue; in its first month on the NSC BBS there were over 400 downloads. Activity on CompuServe, including forum discussions with the authors, has also been busy.

AND MORE OFTEN

You said you'd like to see us more often, so in July 1993 we're going to a bi-monthly publication cycle; you'll receive six copies a year instead of four. While each magazine will be slightly smaller, over a year you'll get the same amount of information in a timelier manner.

SUBSCRIPTION PROBLEMS?

Our publisher, Miller Freeman, has joined us on CompuServe, answering questions about

subscriptions. They also have an Internet ID; you can contact us that way as well.

ORDERING IBM PUBLICATIONS

We've encouraged our authors to list reference publications at the end of each article, many of which can be ordered only through IBM's Mechanicsburg distribution center. To get these publications, call (800) 879-2755. You'll need the IBM order number for each one and a major credit card for billing. IBM employees should continue to order publications through the PUBORDER program on VM.



Dick Conklin

Contact

Dick Conklin
(Editorial)

Miller Freeman
(Circulation)

CompuServe

76711,1005 or
OS2DF2 Forum

71572,341

Internet

OS2MAG
@VNET.IBM.COM

71572.341
@COMPUSERVE.COM

A WEIGHTY TOME

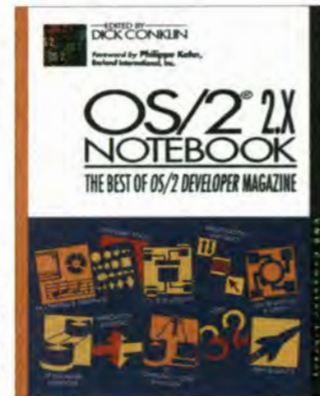
You asked for it, and here it is! The 1100-page *OS/2 2.x Notebook* contains the best of *OS/2 Developer*, 1991-92. It's available at your local bookstore, or call (800) 842-3636 to order.

FOCUS ON CONNECTIVITY

Finally, our theme for this issue is connectivity. Bill Halterman and Don Richards' article covers remote installation and configuration, Julie King and Charles Green weigh in with a piece on extending application interfaces on client/server systems, and Greg Loten addresses issues of connectivity and communication. We top off the issue with Ray Voigt's Spotlight article on Oracle Corp. Enjoy the issue!

Dick Conklin

Dick Conklin





Spotlight

For this issue's Spotlight article, OS/2 Developer visited Oracle Corp., the international developer and manufacturer of the ORACLE server for OS/2, located in Redwood Shores, Calif. **BY RAY VOIGT**

Oracle: OS/2 Preferences and Portability

FOR THIS ISSUE'S SPOTLIGHT ARTICLE, OS/2 Developer visited Oracle Corp., the international developer and manufacturer of the ORACLE Server™ for OS/2, located in Redwood Shores, Calif. Oracle was an OS/2 pioneer with its development of its server for OS/2; in 1992 the company produced the first 32-bit database server to support OS/2. Oracle offers its products, support, education, consulting, and system integration services in 92 countries.

Oracle tries to make application development on a network work as smoothly as it would on a single computer.

Oracle offers an integrated family of portable software, including application packages for accounting and manufacturing, the cooperative server database, and tools for computer-aided software engineering (CASE), application development, and office automation.

ORACLE BACKGROUND

Oracle software products are designed to run on PCs, workstations, minicomputers,

mainframes, and massively parallel computers. The company emphasizes portability and cooperative server technology and tries to make application development on a network work as smoothly as it would on a single computer.

Oracle has consistently led the industry in many areas. First delivered in 1979, the unnamed Oracle database was the first relational database and the first to implement the Structured Query Language, or SQL, which has become an industry standard. By 1983, the product was ported to mainframes, minicomputers, and PCs, making it the first portable database software. In 1991 it became the first database to perform 1,000 transactions per second on the industry standard TPC-B benchmark.

In 1992, the company made another breakthrough when it announced ORACLE7™, the world's first cooperative-server database. A cooperative-server database hides the complexity of a computer network by allowing applications access to data located on multiple computers as though it were stored on a single computer. ORACLE7 supports a large number of users and provides server-enforced integrity, distributed database, security management, query optimization, database administration, and standards compliance.

ORACLE7 enabled applications to retrieve and update this data, making access



to network-based information much easier.

To complement its technical strengths, Oracle emphasizes customer service. CEO Lawrence Ellison notes that the company's ultimate goal is to produce "zero defect software." Until that goal is reached, he has pledged to fix all critical software defects within 24 hours.

INTRODUCTION OF THE ORACLE SERVER FOR OS/2



Romeo Baldeviso

Romeo Baldeviso, product line manager for the ORACLE Server for OS/2, serves as a contact with other companies and development organizations, whose requests and suggestions he helps implement at Oracle.

Oracle's commitment to OS/2 was strengthened by OS/2's success overseas. "I've seen the market abroad take off," said Baldeviso. "European sales are very strong and growing." The European market, adds international marketing manager Elizabeth Donahue, is much more aggressive, leading the non-U.S. market by six months to a year in adopting new technologies. With the announcement of the ORACLE Server for OS/2 in 1989, explains business development manager Anita Planenshek, Oracle supported OS/2 "with a lot of encouragement and education. It's paid off; we were able to have an Oracle product running on OS/2 2.0 at [OS/2's] release."

Oracle's server development process for OS/2 starts with base products operating on larger systems. Baldeviso explains that an Oracle base group first develops a product on UNIX or VMS.



Anita Planenshek

The desktop group then takes over and ports the code to OS/2 2.0, writing and modifying an operating system-dependent (OSD) layer, optimizing it for the OS/2 environment.

The company's goal is to produce "zero defect software." Until that goal is reached, CEO Lawrence Ellison has pledged to fix all critical software defects within 24 hours.

PORTABLE AND SCALABLE SERVERS AND DISTRIBUTED DATABASES

The ORACLE Server for OS/2 contains the RDBMS, which runs on over 85 hardware platforms. Its scalability across systems allows for structured client/server database systems with low-cost PC servers that can be added to existing local area networks. The server's portability also allows customers to expand client/server systems by migrating data upward to more powerful processors such as minicomputers or mainframes. Finally, the server's adaptability allows flexibility during distribution, load processing, and system response maintenance while minimizing hardware and training expenses.



Elizabeth Donahue

ORACLE'S VIEWS ON PERFORMANCE

The ORACLE Server is very sophisticated; originally created for use on VMS multiprocess machines and large mainframes, it already had much of the sophistication

required by OS/2 2.0. The company's product managers decided to base their desktop application on a 32-bit operating system. Comments Baldeviso, "There wasn't a lot of convincing required. OS/2 is available now, it [is] shipping now, in production now." Oracle's decision was also influenced by a 60% performance improvement from the 16-bit to the 32-bit versions of OS/2.

Larger Workgroups. ORACLE Server's performance and number of supported users are limited by the amount of RAM available in the server. With more RAM, a larger area is available as a database cache. This increases the chance that a data request will result in a "direct cache hit" (direct RAM access), rather than the much slower disk storage access. With more RAM available, the transaction rate increases and the response time decreases.

The server RAM is also needed for the Oracle client shadow process. Each client process

requires approximately 250 KB per connection, depending on the application used. With OS/2's additional RAM support, more users can connect to the server at one time.

Ease of Use for Application Development Tools. ORACLE Server users can choose from several Oracle development tools, including CASE products and fourth-generation language tools such as

SQL*Forms™ and SQL*Menu™. Also available are graphic tools for Windows™ such as Oracle Card for Windows™ and Oracle for Windows™, as well as development tools from independent vendors.

Oracle's CASE products for OS/2 support the Workplace Shell. Because Oracle products are built for deployment in client/server configurations and heterogeneous networked environments, users can distribute and share development resources across platforms. CASE* Generators™ produce applications that are portable to over 100 platforms and can fully exploit client/server configurations. They can reside on an OS/2-based PC or on other platforms.

Because Oracle's CASE products are available on OS/2 2.0, it is easy to construct inexpensive and flexible application development configurations. For example, systems analysts with OS/2-based workstations can use CASE*

```

File Edit Session Instance Storage Log Backup Security Monitor Help
                                Output
                                ORACLE Session Statistics Monitor

Starting Session Id: 0          Ending Session Id: 99999

[ X ] User      [ ] Redo      [ ] Enqueue    [ ] Cache    [ ] Debug

Statistic Name                                Current Average Minimum Maximum Total
-----
CPU used by this session                      0         0         0         0         0
CPU used when call started                    0         0         0         0         0
background timeouts                           4         4         2         6        2480
cumulative opened cursors                     2         .5         0         2         384
current logons                                0         0         0         0         7
current open cursors                          0         0         0         0         1
cursor authentications                        2         .5         0         2         700
logical reads per session                     29        7.25       0        29        3922
logons                                         0         0         0         0         13

                                (Restart) (Hide) (Quit)

```

ORACLE7 Statistics Monitor

```
File Edit Session Instance Storage Log Backup Security Monitor Help
                                Output
                                ORACLE Table Access Monitor

Minimum Session ID: 0          Schema Filter: %%
Maximum Session ID: 999999    Table Filter: %%

Session
ID      Schema Name          Table Name

7 SYS          U$ACCESS
7 SYS          X$KGLDP
7 SYS          X$KGLLK
7 SYS          X$KGLOB
7 SYS          X$KSUSE

                                [Restart] [Hide] [Quit]
```

ORACLE7 Table Access Monitor

Dramatically Reduce Your Development Time



The Natural Migration

IBM offers software developers a unique opportunity to migrate their existing DOS, Windows™, UNIX®, and 16-bit OS/2 applications to the Platform of Choice for the 1990s and beyond: 32-bit OS/2.

The IBM OS/2 32-Bit Application Migration Workshops give developers hands-on, in-depth training and personal assistance in migrating their existing application to OS/2. They minimize the learning curve associated with a new operating system, and they accelerate migration efforts by providing industry experts who instruct and assist in the migration process.

The following is a list of OS/2 32-Bit Application Workshops for 1993 now being offered:

Windows 3.x OS/2 32-Bit PM Native Workshop

DOS to OS/2 32-Bit PM Workshop

System Object Model/Workplace Shell Workshop

OS/2 16-Bit PM to OS/2 32-Bit PM Workshop

UNIX to OS/2 32-Bit Workshop

For more information, and to enroll, call One Up Corporation and refer to the IBM OS/2 Application Migration Workshops:

1-800-678-31UP

Reserve your place in the OS/2 32-Bit Application Workshop now being offered in Austin, Texas:

OS/2 Distributed Computing Environment (DCE) for Software Developers

For more information, and to enroll, call:

1-800-IBM-TEACH

Designer™. CASE*Dictionary™ tables, which store and control access to application development information, can reside on a stand-alone OS/2-based PC or on multiple platforms in a client/server and multiuser configuration. Information sharing between platforms is accomplished in real time by SQL*NET™. Other configurations are possible with a variety of IBM and non-IBM platforms.

Oracle's mainframe and mid-range experience "seems to give people a little better preparation for a 32-bit system like OS/2."

SQL*NET 1.1

The new LAN SQL*Net SPX™ 1.1 and SQL*NetBIOS™ 1.1 are part of the ORACLE Server. The SQL*Net 1.1 drivers prespawn listening processes, generating faster connections to the server. Instead of waiting for the client connection request before spawning a user process on the server, SQL*Net 1.1 prespawns a tuneable number of processes, speeding up client connections. A new user interface displays the status of the SQL*Net listener, showing the number of current connections as well as the number of prespawed processes awaiting connections.

Additional wide area network connectivity is provided through Oracle's SQL*Net TCP/IP™, SQL*NetDECnet™, and SQL*Net APPC™ (all are available independent of the server). This soft-

ware allows the server to access distributed data from a variety of other Oracle platforms including DEC, VAX, IBM mainframes, UNIX machines, and NetWare file servers.

SQL*NET, Oracle's networking product within the desktop, actually sits on top of the network's transport layer. Oracle allows a tool, such as a graphical user interface or query tool, to be physically located on a separate machine across a network. SQL*NET packages the queries on the tool side and sends them over to the SQL*NET on the server side, which pulls the data off the network and sends it to the server. Katrina Montinola, group development manager for SQL*NET, explains, "The tool has no idea that it's actually talking to a remote server because of this thin layer of software. The server, meanwhile, has no idea that the tool is across the network."

32-BIT APPLICATION BUILDING

OS/2 Preferences and Portability.

Because of the power and ease of use of OS/2, most programmers in the company's desktop production division do their development work on it. While many smaller companies are upgrading to OS/2 from DOS environments, Oracle's mainframe and mid-range experience, says Montinola, "seems to give people a little better preparation for a 32-bit system like OS/2." Among Oracle programmers, she continues, "some people relate to it better than others because of



Katrina Montinola

their background;" most of the company's programmers are recent graduates with experience in UNIX-based environments. Regardless of individual background, however, the desktop products division is very pro-OS/2: "We use OS/2 as our development environment no matter which product we're developing—DOS, Windows or OS/2. Of the seven operating systems that we support, I have to say that OS/2 is the easiest to support."

In general, the desktop product developers found OS/2 significantly easier to use than UNIX, despite the time required to master components unique to OS/2, such as Presentation Manager. There is no formal OS/2 training at Oracle, but Montinola says that her situation is common: "I never actually learned DOS; I went straight from a UNIX-based school to IBM using OS/2....It didn't seem that different."

Compatibility and portability are always important to Oracle's programmers, no matter which operating system is used. Explains Montinola, "There are problems when you try to write base code on an operating system more powerful than that you're porting to," as with UNIX to DOS. But there is little problem with OS/2: "It's unfair to say it's really easy to port from UNIX to OS/2, because our situation is unique due to how Oracle designs its software.... Oracle developers always have porting in the back of their minds when they write a program."

Besides a general awareness of portability requirements, Oracle maintains a set of well-tested coding standards. Almost everything is written in C. All files that can be considered "operating system dependent"—such as file access or screen access files—are localized. Although the situation is slightly different for each operating sys-

GET ON A CLIENT/SERVER TRACK WITHOUT DERAILING YOUR DEVELOPMENT

TO PUT YOUR APPLICATION DEVELOPMENT ON A CLIENT/SERVER TRACK FAST, USE THE TOOLS THAT GET YOU WHERE YOU'RE GOING WITHOUT SETTING YOU BACK. KASEWORKS PROVIDES A FULL SUITE OF VISUAL DESIGN AND CODE GENERATION TOOLS THAT TRANSPORT YOUR MAINFRAME, MINICOMPUTER AND PC-BASED APPLICATIONS INTO AN OS/2™ GUI-BASED CLIENT/SERVER FUTURE WITHOUT ABANDONING YOUR CURRENT DEVELOPMENT ASSETS.

BECAUSE WE MAKE MOVING FROM PROCEDURAL TO EVENT-DRIVEN DEVELOPMENT EASY, WE PROTECT YOUR INVESTMENT IN EXISTING DEVELOPMENT EXPERTISE. WE GENERATE APPLICATIONS IN STANDARD LANGUAGES LIKE C, C++ AND COBOL, SO WE PROTECT YOUR INVESTMENT IN EXISTING APPLICATIONS. AND SINCE THERE'S NOTHING PROPRIETARY ABOUT OUR ENVIRONMENT, YOU CAN DEVELOP WITHOUT RESTRICTIONS—YOU OWN THE GENERATED CODE, WITHOUT RUNTIMES OR ROYALTIES.

UNLIKE OTHER TOOL COMPANIES, KASEWORKS GIVES YOU COMPLETE CONTROL OVER YOUR DEVELOPMENT DIRECTION BY LETTING YOU LICENSE OUR PATENTED TECHNOLOGY TO CUSTOMIZE OUR TOOLS TO YOUR NEEDS. YOU CAN PROTOTYPE AND GENERATE SPECIFIC LINE-OF-BUSINESS FUNCTIONS, PROPRIETARY APIs, SPECIALIZED INTERFACE STANDARDS, UNIQUE DATABASE "HOOKS" AND OTHER REQUIREMENTS. ALL OF WHICH LETS YOU LEVERAGE THE EXPERTISE OF A FEW DEVELOPERS ACROSS YOUR ENTIRE ORGANIZATION.

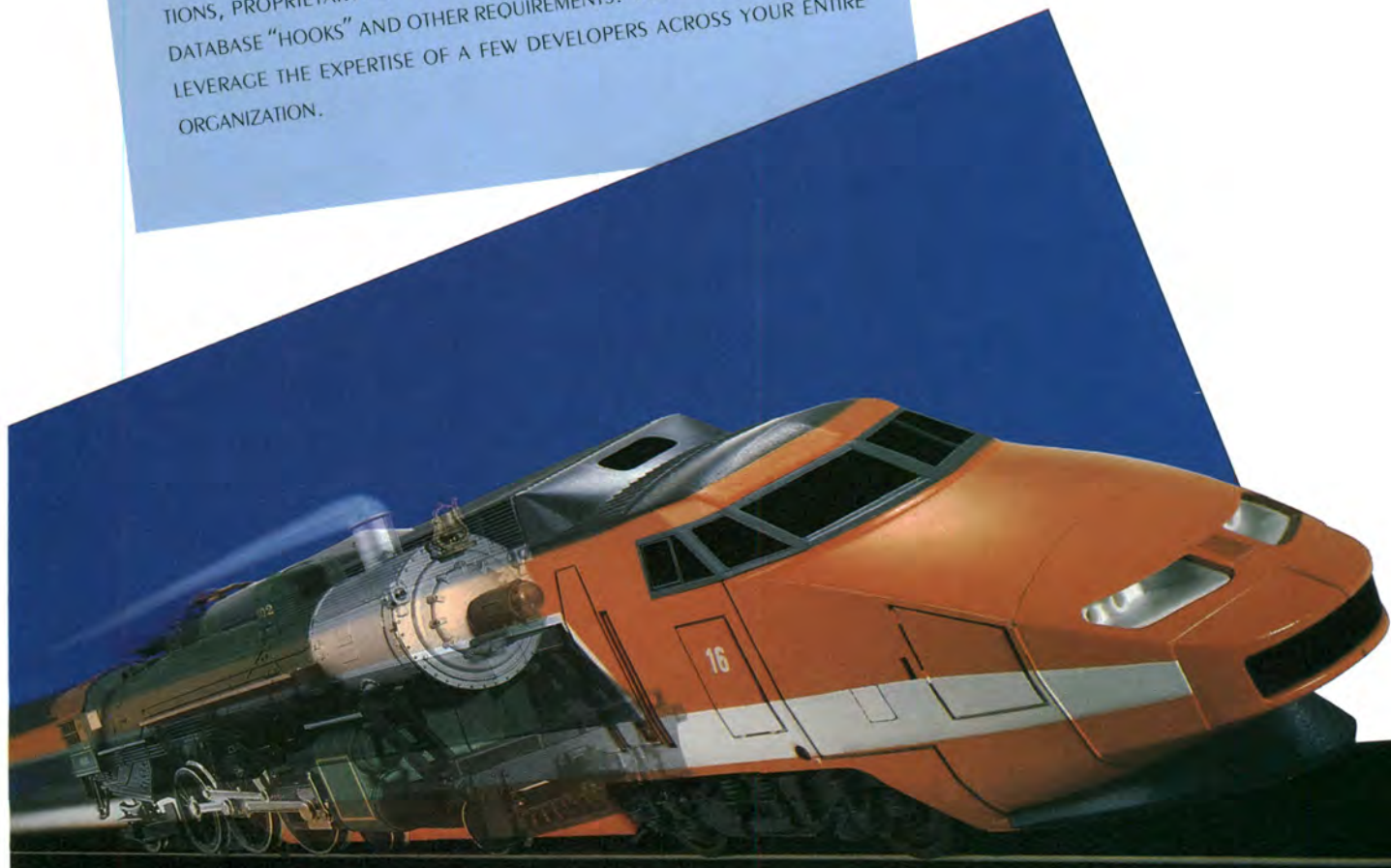
GET ON A CLIENT/SERVER TRACK WITHOUT BLOWING A LOT OF STEAM — OR RESOURCES. CALL US TODAY AT 1-800-635-1577 FOR A FREE DEMONSTRATION DISK OF OUR CASE:VIP TOOLS, OR FOR MORE INFORMATION ABOUT KASEWORKS DEVELOPMENT SEMINARS IN YOUR AREA.

KASEWORKS™

△△△△ Enabling Client/Server Strategies

3295 RIVER EXCHANGE DRIVE, SUITE 430
NORCROSS, GA 30092

(404) 448-4240 • (800) 888-4335
FAX (404) 448-4163



tem, with approximately 80% of the code "we just compile the file and it's fine," says Montinola. Most changes are made to the file system, the layer on top of a network protocol stack, and similar parts unique to an individual system.

Colello: "It isn't mandatory, but about four years ago somebody started using it and it caught on. Now it's convenient because everybody knows, when they jump from machine to machine, which tools are where, and the environment is familiar. Some of

Parallel Development. When it comes to parallel development, developers identify those proportionally few files that are system-dependent and port them for each individual platform; the rest run on many different platforms and need no adaptation. "In general," says Colello, "a product like ORACLE7 has maybe 1000 server files; maybe 50 to 100 are system-dependent, so the other 900 you never touch. For instance, on the ORACLE7 Server, I build products for OS/2 2.0, OS/2 1.x, DOS, and Windows using the same source files. Each product has different make files, but the source files are basically similar, with different compile options and compilers."

Colello is a big fan of OS/2, even when programming for other systems: "I've been here three and a half years, and I've never actually built a DOS product in DOS or a Windows product in Windows. I always build in OS/2 and reboot."

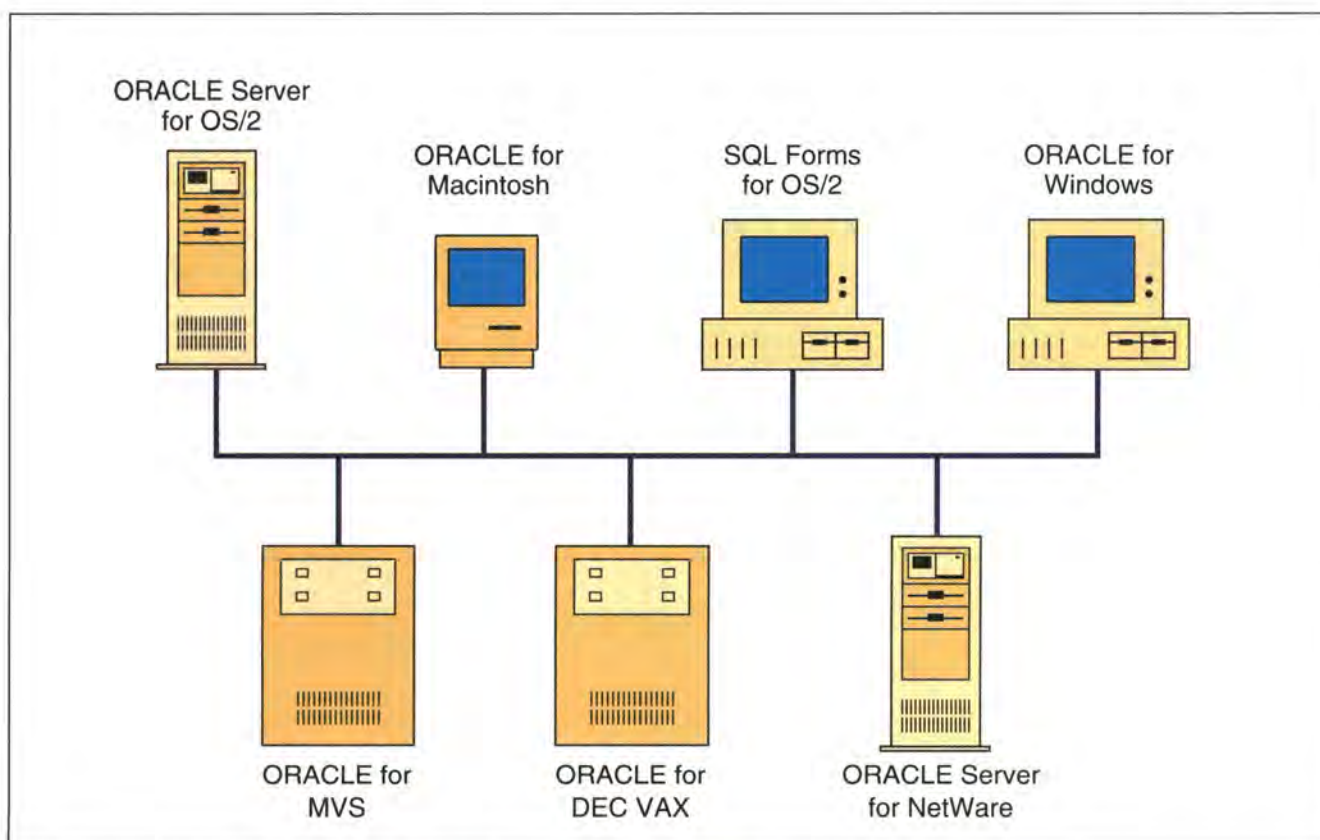
All files that can be considered "operating system dependent"—such as file access or screen access files—are localized.

DEVELOPING THE ORACLE SERVER FOR OS/2

Tools and Tips. Over the years, the Oracle developers have established common habits and tools that help them to coordinate their work. The KEDIT™ editor is popular, says senior developer Dave

the people with UNIX backgrounds use Epsilon and EMACS, but there's no real standard."

Automated test tools developed internally at Oracle are also common. For development tools, the programmers rely on the IBM 32-bit C Set/2™ compiler and the OS/2 Toolkit.



The ORACLE7 database can run multiple systems for enterprise-wide connectivity

He gestures toward his computer. "Under OS/2 you get your protection exception that tells you what happened.... under DOS or Windows, you hang the machine, and then you have to search for the answer. In that respect, it's just easier to use OS/2. It makes so much more sense."



Dave Colello

While most Oracle developers come from a large systems background, they find OS/2 comfortable to work with. Several mentioned the operating system's sophistication and power, likening

OS/2's multitasking to that of a mainframe. Senior developer Pete Sciarra, who once worked exclusively on mainframes, especially appreciates OS/2's interaction with REXX, a language commonly used on VM and other large operating systems. Says Sciarra, "As more people use REXX and start to see its capabilities, it'll become a very powerful tool and a big piece of OS/2.... people will be able to use it across all of their linked systems."

Using OS/2 Features For Smooth Porting. Several developers took advantage of OS/2's unique features such as semaphores, threads, and shared memory when designing ORACLE7. As a multiprocessing product, it had to communicate with other running processes; the developers used semaphores to synchronize access to global vari-



Pete Sciarra

ables and similar objects. They also used shared data and threads to process information along to theme variables, speeding up data flow. Notes Colello, "DLLs also play a big role in our operation. We have many processes, and being able to share code among all the processes is a major win for us. It lets you choose whether to use shared memory. It lets you use DLLs and load code on the fly. And it lets you run multiple processes....in a friendlier environment, it's easier to port code."

Development manager Van Okamura, whose group was responsible for porting ORACLE7 to OS/2, says the process was unusually smooth. "We started to port early and worked with IBM to get a lot of the compiler and operating system problems resolved before we went to production. I think both companies benefited from this type of feedback." It also helped that Oracle already had a

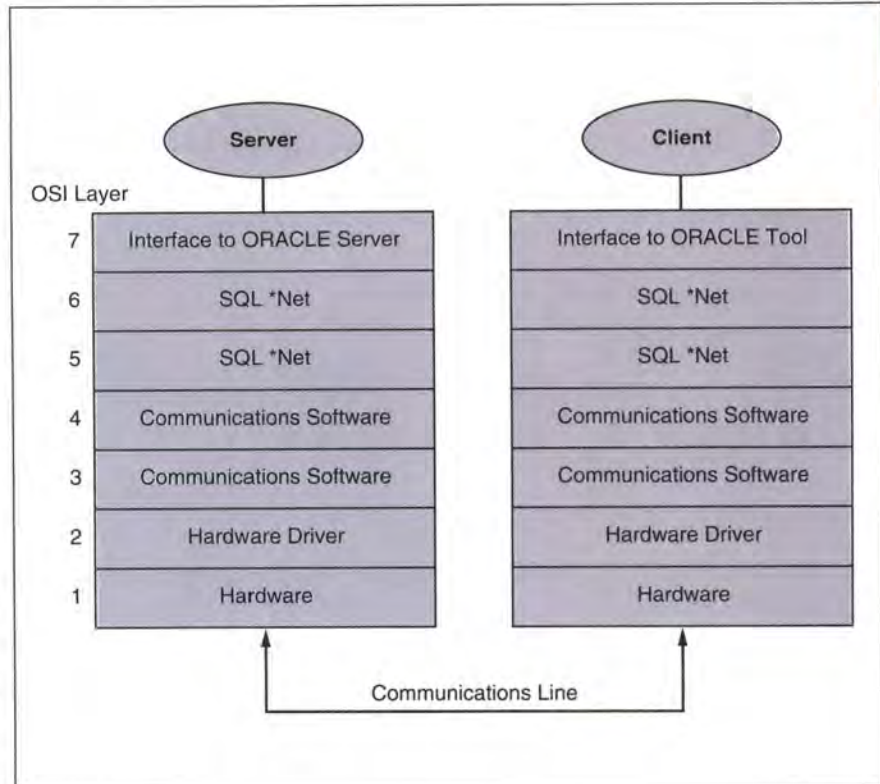
database server for 16-bit OS/2 and had only to port the code to make it unique for the 32-bit system.

Emphasis was also placed on making Oracle applications easy to install and use. In one example, Oracle developed a portable installer that runs on OS/2, DOS, UNIX, and Windows: "The installer reads the script files, and the installation looks the same for all of the platforms," says Colello. Montinola found OS/2 easier to use when working on SQL*

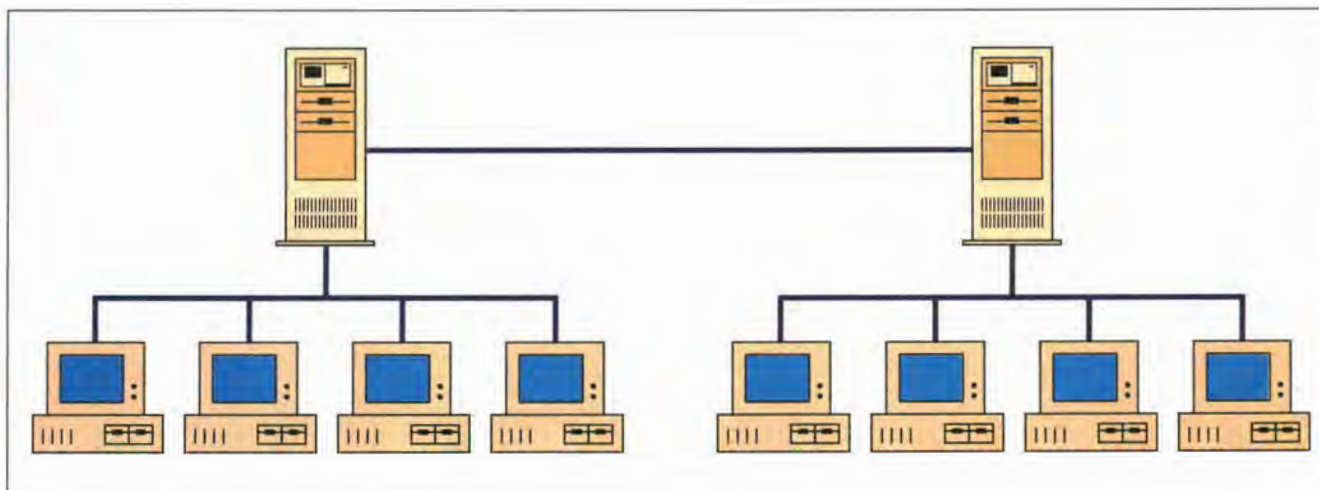


Van Okamura

Net: "It's easier than working in the UNIX world, for example, when a customer has to link SQL*Net to a tool or to the server on a site. Because SQL*Net for



SQL *Net interface



Second generation cooperative-server database

OS/2 is implemented as a DLL, it's dynamically linked...you don't have to worry about on site linking like you would with UNIX."

"We have so many processes, and being able to share code among all the processes is a major win for us."

Beta Testing and Feedback. Oracle's beta test process is organized to easily track and evaluate problems. Selected customer sites participate, says Baldeviso, with help from Oracle's worldwide support group. Testers file the bugs and they are addressed as they come in. A small group of customers known as "desktop internal contacts" also take the beta software. Once a product is on the market,

the customer input doesn't end. An Oracle forum on CompuServe, a worldwide customer support line for product owners, and Internet™ access keep the discussion lively.

NEW FEATURES IN THE ORACLE SERVER

The ORACLE Server for OS/2 2.0 incorporates a number of new components for improved performance, including new data loading facilities, National Language Support (NLS), and a C precompiler for building 32-bit applications. The server supports 18 European languages for reporting status and error messages.

THE FUTURE: THE ORACLE 7 SERVER AND OS/2 2.0

The ORACLE7 server for OS/2 2.0 introduces the cooperative-server database, which enables applications to access data located on several computers as if it were stored on a single computer. While early client/server databases require extra programming to access data on more than one server, ORACLE7 supports industry standards for

SQL query and update transactions that automatically retrieve and modify data on multiple servers. A cooperative-server database also allows a group of low-cost server computers to outperform much larger mainframes. Finally, the database offers high reliability because there is no single point of failure. The database also provides advanced distributed database capabilities that make remote database access transparent to the user, eliminating the need for transaction updates. Finally, the server provides new functions such as stored triggers and procedures, server-enforced integrity, and security management.

Ray Voigt, IBM Corp., 1000 N.W. 51st St., Boca Raton, Fla. 33431. Voigt has worked for IBM for over 16 years, with jobs in field service and information development. He has worked on the OS/2, DOS 4.0 and 6.0, and multimedia libraries, and is currently a service planner. He holds a M.A. in instructional design and audiovisual education and a B.S. in industrial education, both from Eastern Illinois University, Charleston, Ill.

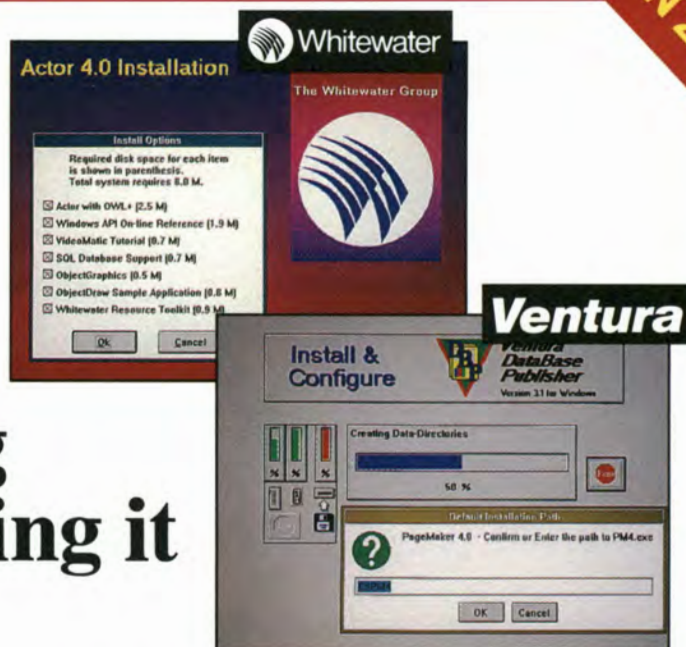


A SHIELD Series
Development Tool

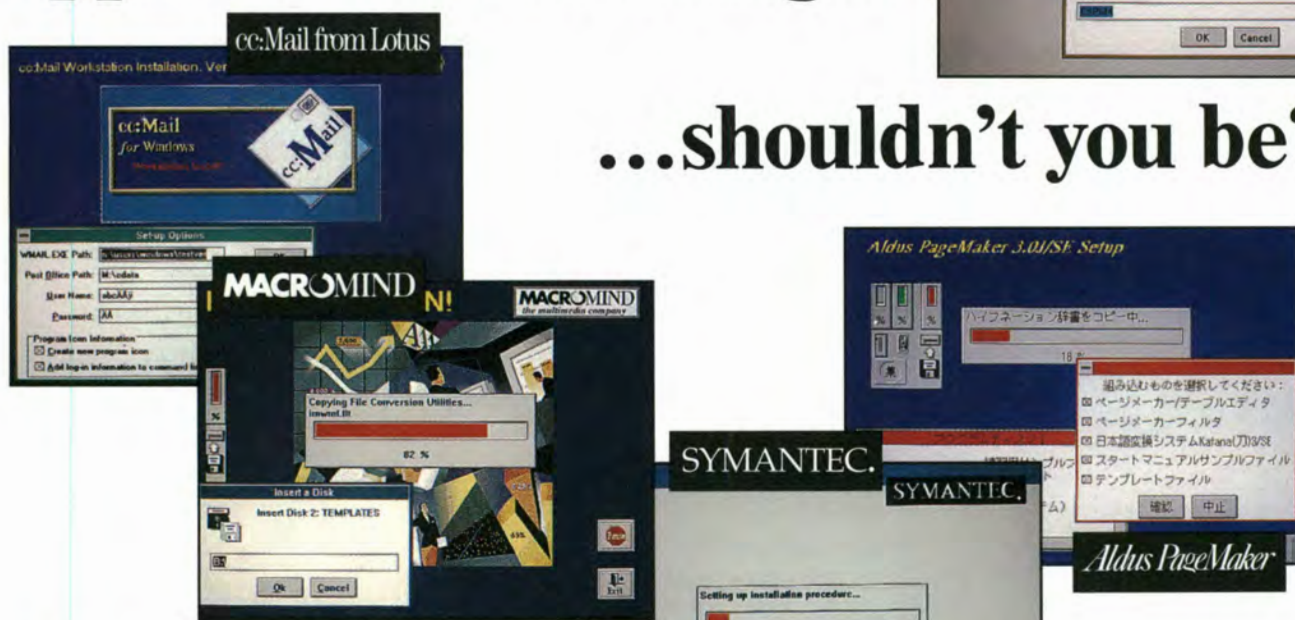
NEW
VERSION 2.0

InstallSHIELD™

The world's leading
applications are using it



...shouldn't you be?



First impressions last!

Use InstallSHIELD to create a professional, bullet-proof installation program for your application. Full professional features, easy to use. Unconditionally guaranteed. No royalties.

In
Version
2.0

- Full Support for Windows 3.1
- Full Support for OS/2 2.0
- Over 200 New Features
- New! Easy 10 Minute Install Builder



Product and company names are trademarks of their respective holders.



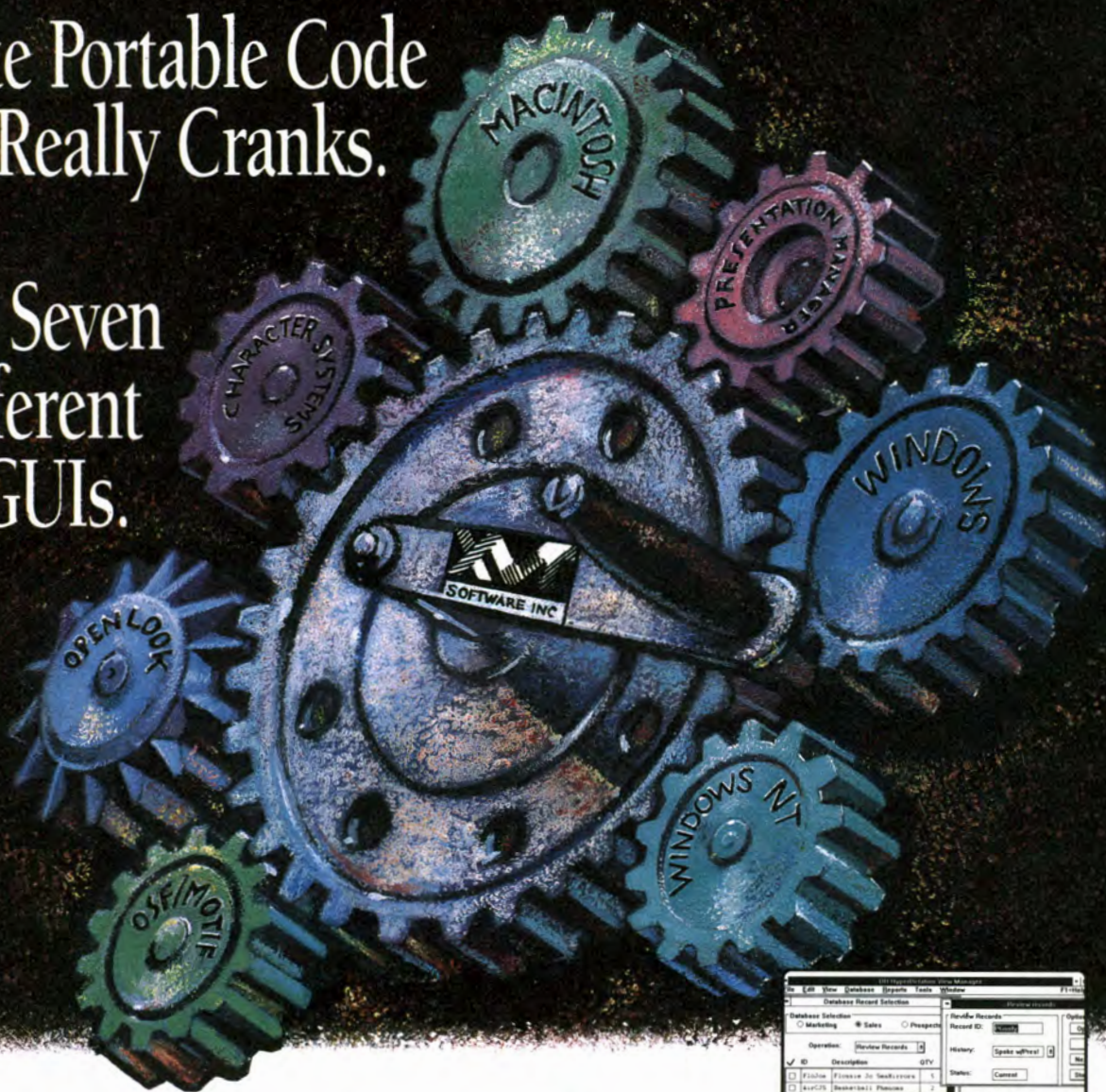
The Stirling Group®
Making it easy to make™

172 OLD MILL DRIVE • SCHAUMBURG, IL 60193 • USA • CALL 708-307-9197 • FAX 708-307-9340 • CompuServe: GO STIRLING

1-800-3 SHIELD
1-800-374-4353

Write Portable Code That Really Cranks.

On Seven Different GUIs.



XVT's Portability Toolkit™ is a powerful C development environment that allows you to build a single application, then re-compile to every major GUI without rewriting code. XVT solutions also include an interactive design tool and a class library for C++ developers.

- Supports Macintosh, Microsoft Windows, Windows NT, OS/2 Presentation Manager, OPEN LOOK, OSF/Motif, and Character Systems
- Native look-and-feel to all target GUIs
- Portability to 26 hardware systems
- Access to the complete functionality of every windowing system
- Easier to use than native development toolkits
- Minimal size and performance overhead
- Shorter development cycles
- No royalties or runtime fees
- Clear documentation and responsive technical support

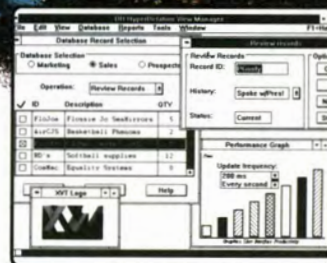
Now in its third generation, XVT is recognized as the industry leader in portable GUI development solutions and is the base document for the emerging IEEE standard. It is used by world-class software developers like •Novell •HP •AT&T •Digital •Lockheed •Kodak •Grammatik/Reference Software, because it allows them to take their applications to the widest market, quickly and cost effectively.

Don't write another line of code without gearing up to develop your application simultaneously for all GUIs. Call for technical materials and a demo.

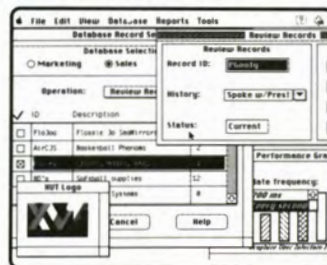


The portable GUI development solution.
1-800-678-7988

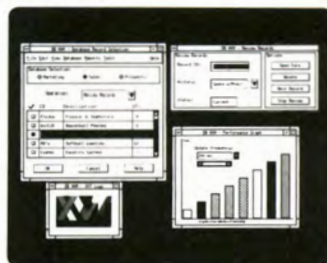
XVT Software Inc. 4900 Pearl East Cir. Boulder, CO 80301
(303) 443-4223 FAX (303) 443-0969
For European inquiries, contact: PVI Precision Software GmbH
Phone: 49 0 61 03/37 94 0 Fax: 49 0 61 03/36 95 5



▲ Microsoft Windows & Windows NT



▲ Macintosh



▲ OSF/Motif

Shown above are four of the seven GUIs supported by XVT.

XVT and XVT Portability Toolkit are trademarks of XVT Software Inc. Other product and company names are trademarks or registered trademarks of their respective holders.



This article describes how Communications Manager/2 1.0 implemented configuration, installation, and distribution techniques to support its remote installation and configuration. BY BILL HALTERMAN and DON RICHARDS

Automating Application Installability With Communications Manager/2

THIS ARTICLE DESCRIBES HOW Communications Manager/2™ 1.0 implemented configuration, installation, and distribution techniques to support its remote installation and configuration. It also covers how application programs can use these techniques to query and update Communications Manager/2 installation and configuration data and how they can be applied to applications to allow installation and configuration without user interaction.

INSTALLATION ENVIRONMENT

Customers install OS/2 workstations in several ways; many companies preload new workstations at a central site, generally right out of the box. A "sneakernet" method, in which applications are installed on every workstation from disks, is also common. Some companies use cloning methods to copy product files from one workstation to another; values unique for each user are specified at the workstation. To upgrade workstation software, customers may mail hard drives between locations, send technical experts, or hire outside vendors to perform this laborious task.

These traditional methods, however, are neither efficient nor cost-effective, and large customers now demand a simple way to install and configure multiple OS/2 workstations. To solve this problem, IBM developed a methodology known as configuration, installation, and distribution, or CID, which has

been implemented in OS/2 2.0, various products from other vendors, and Communications Manager/2 1.0. This release, the successor to OS/2 Extended Services Communications Manager, runs on OS/2 2.0 and higher as well as on OS/2 1.3.1 with Corrective Service Diskette 5050 or higher.

OVERVIEW OF CID TECHNIQUES

The first step to implement CID in Communications Manager/2 was to eliminate the need for floppy disk-based installation. The installation program CMSETUP can now install from a source drive other than the A: drive. The CMIMAGE utility copies the Communications Manager/2 image files from the disks into a subdirectory on the hard disk. Rather than install the product, it copies the compressed files to the hard disk as they appear on the floppy disk. At installation, the files are then unpacked to the target drive. This process, called redirected installation, works well in a LAN environment in which users do their own installation but wish to speed up the process by not handling floppy disks. Redirected installation is diagrammed in Figure 1.

Despite Communications Manager/2's redesigned installation and configuration user interface, the installation process may be too labor-intensive for administrators and too complex for novice users. One solution is to control this procedure through a stored set of responses to user questions asked during the installation process. Using this stored set of



Bill Halterman



Don Richards

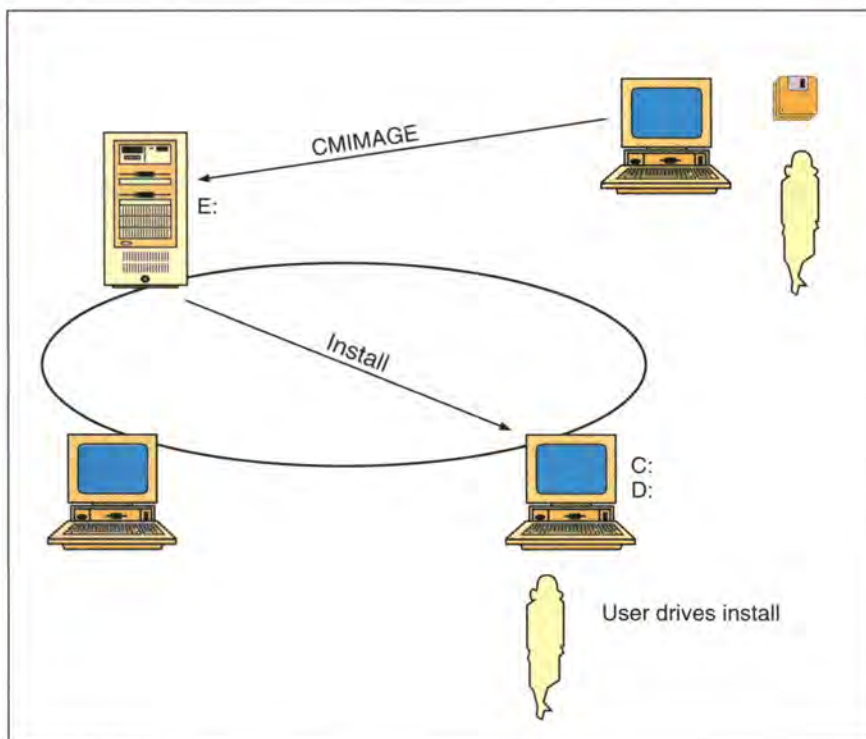


Figure 1. Redirected installation

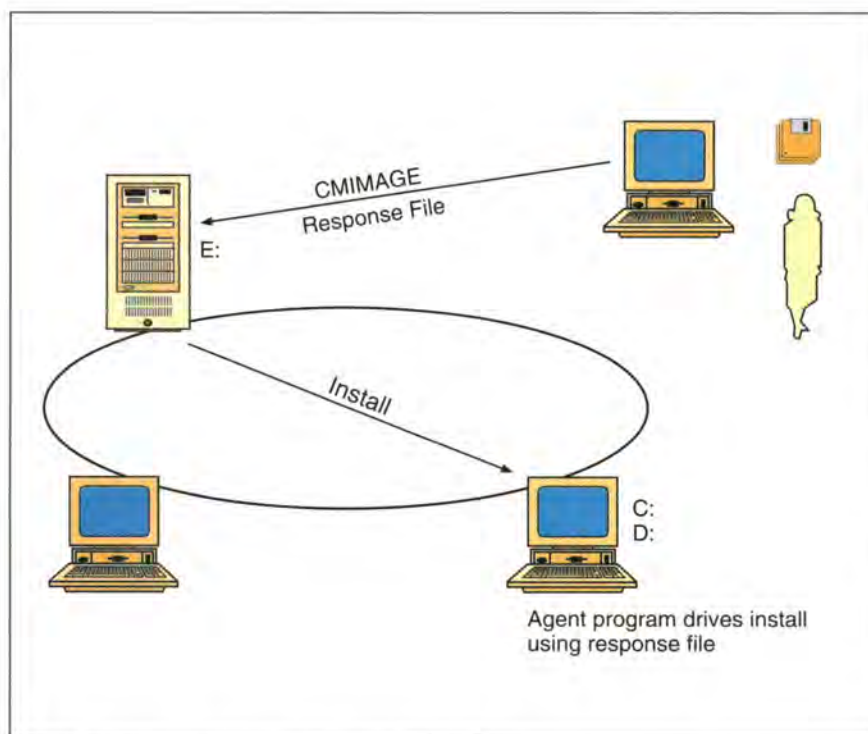


Figure 2. Unattended installation using response files

responses, called a response file, avoids the process of traversing the installation and configuration windows. Response files are ASCII-for-

mat files containing entries in a key-word = value format. For example, `CMTarget = D:` is specified to select the drive used to install the product.

When installing an application from a response file, minimal user interaction is necessary. This is called a "lightly attended" installation. For completely unattended installation, a distribution product such as IBM NetView Distribution Manager/2™ 2.0 is used. In this case, the installation process is run by an agent program without a user present, as shown in Figure 2.

In addition to redirected access to install images and response file support, Communications Manager/2 supports other CID techniques:

- Generating a response file from an installed Communications Manager/2 system
- Implementing CID-defined command-line parameters and return code values for the installation program
- Recording errors during installation to a log file and copying the files to a server location
- Supporting one or more user exits during installation.

MANIPULATING COMMUNICATIONS MANAGER/2 CONFIGURATION FROM AN APPLICATION

Querying installation and configuration values. Since a good portion of the Communications Manager/2 configuration is in a binary file, an ASCII equivalent is helpful for viewing or manipulating the configuration values. The `CMRECORD` utility takes a configuration file as input and generates a response file that includes installation and configuration keywords for the workstation on which the utility was executed. An application can invoke `CMRECORD`, process the results, and provide a customized interface that allows users to change values or delete entries. A system administrator can also invoke `CMRECORD` on an already installed workstation, modifying

the response file to install on multiple workstations, if necessary.

To run CMRECORD, issue the command `CMRECORD /D /O C:\MYDIR\MY-APPL.RSP` from your application on an OS/2 workstation where Communications Manager/2 is installed. This action generates a response file, using the default configuration file for that workstation. The response file contains keywords representing the contents of the configuration file and relevant installation response file keywords. The output file is stored in MYAPPL.RSP. Since Communications Manager/2 can use multiple configurations on a single workstation, a different configuration file can be specified by providing the file name in place of the /D parameter.

There are several other options available with CMRECORD. The generated response file can output every configurable field that contains a

value, including defaults, for the entire configuration file. The amount of output can be reduced by providing a model response file as input (using the /M parameter), containing only the keywords for which you want to query the current value. Another way to limit the output in the generated response file is to use the /K parameter and specify the keywords for which you want to obtain a value.

Building a response file. As an alternative to CMRECORD, the Communications Manager/2 response file can be created from scratch with an ASCII editor. The syntax is documented in the *IBM Communications Manager/2 Version 1.0 Network Administration and Subsystem Management Guide*, listed in the References section at the end of this article. (This publication also includes a disk containing sample response

file templates that help with tasks such as upgrading an existing workstation from Extended Services or configuring 3270 emulation over Token Ring for a new workstation.)



Another way to limit the output in the generated response file is to use the /K parameter and specify the keywords for which you want to obtain a value.

Using A Model Configuration File. A useful feature of Communications

Tennis Pros, Golf Pros And OS/2 PM Programmers Have One Thing In Common...

...they need proper training and tools to be successful !

The right mix makes all the difference! PM programming is easier than you think **IF** you use proper object-oriented techniques and high level PM window classes.

Ask for **SE International's**

- ✓ **Inhouse PM programming training.** Within one week you learn how to succeed in large C and PM projects and how to choose the right development tools.
- ✓ **PM Extensions.** Powerful new 16-bit and 32-bit PM window classes (**FormattedEntryFields**, **DateEntryFields**, **PrimaryWindowClass**, **Animated Bitmaps**, **MultiColumnListBoxes**...) allow you to focus on the application logic and not on programming of frequently required window behaviors. All our **PM Extensions** are **Gp^f**™ compatible.

SE International, Inc.

621 NW 53rd Street, Boca Raton, FL 33487
Tel: (407) 241-3428 • Fax: (407) 997-8945

SES Software Engineering Services GmbH Berlin

Koenigsallee 49, D-1000 Berlin 33, Germany
Tel: +49 30 826 40 63 • Fax: +49 30 825 30 14

Free OS/2 Utility Disk*

OS/2 Monthly Magazine, **the independent guide to OS/2 computing.** Whether you are New user, a Developer, or a even Windows-OS/2 user, OS/2 Monthly has invaluable information Info. Programming and User Articles, Reviews, Market info, all by OS/2 Experts. With a paid subscription you'll receive the disk and 1 year (12 issue) subscription. (*\$10 value.) Visa, MC, Check, Money order accepted.

☐ **Bill me and start my 1 yr subscription (12 issues)**, when I pay the \$32.99, (OS/2 Developer Subscribers pay \$29.99), I'll get the OS/2 utility disk. (\$45 US in Canada; \$87 US elsewhere.)

Name

Company

Address

City State Zip

Credit Card

Expiry Date

Mail to: JDS Publishing, PO Box 4351, Highland Pk, NJ, 08904

Add'l \$3 Discount for User Groups members or for mentioning the OS/2 Developer

For FASTEST service call 800-365-2642

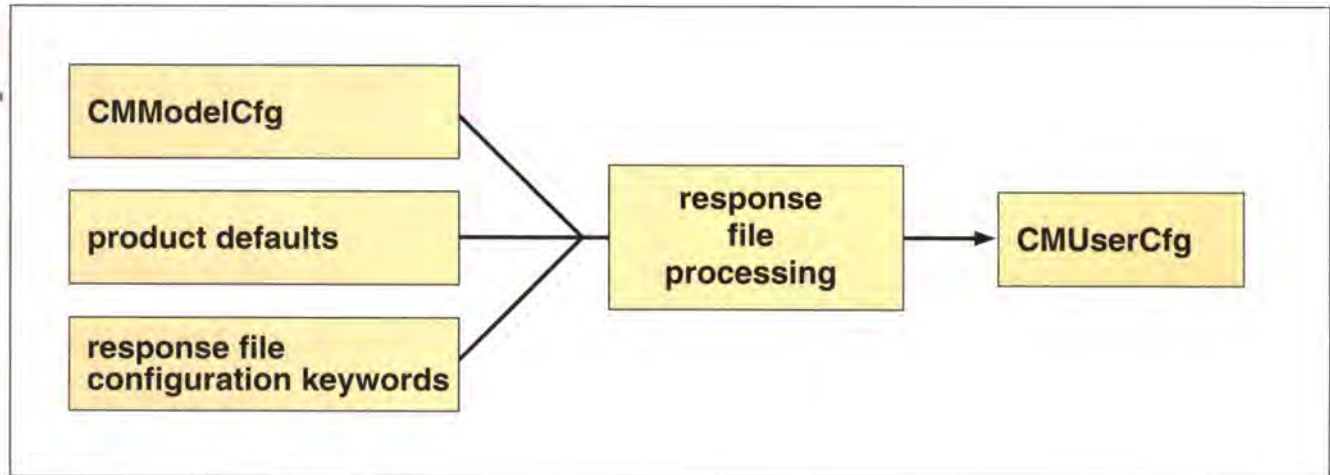


Figure 3. Building a user configuration file

Manager/2 response files is that they need not contain every parameter to be configured. A model configuration file can be built to represent the type of work a group of users perform. For example, the group members each use two 3270 emulation sessions on a Token Ring LAN. One model configuration file can then be used to install each similar workstation in a network. Each workstation uses a small customized response file that references the model configuration file and

contains a significant number of configuration parameters.

Updating Configuration. To begin processing a response file, start the CMSETUP program with the /R option. For example, if the Communications Manager/2 disk images have been loaded into the directory X:\IBM\CM2 and the installation response file is named X:\RSP\NEWINST.RSP, issue the command:

```
X:\IBM\CM2\CMSETUP /R X:\RSP\NEWINST
```

to begin the installation. Two log files are created showing the result of the installation; CMRINST.LOG contains the main response file processing, configuration, and verification messages, while the installation log file CM.LOG contains additional status and error messages. The /L1 and /L2 parameters can be specified to indicate that the log files should be copied to another location, such as an accessed hard drive at a server workstation. A CID-defined return code is also returned for use by the distribution agent. For example, a return code of X'FE 00 indicates to NetView Distribution Manager/2 that the product installation was successful and that rebooting is required to complete the process.

We will now look at some instal-

lation response file keywords, with examples that use the response file-redistributed install process for remote setup of an OS/2 workstation with Communications Manager/2.

Important Response File Keywords.

The response file support in Communications Manager/2 includes more than 700 keywords that perform unique installation and configuration actions. Before giving specific examples of response files in action, we will cover the most important installation keywords. These examples should give an idea of how to define application-enabling keywords.

The keyword CMUpdateType must appear in every response file; it numerically selects the action that the file will perform. For example, a value of 1 for CMUpdateType tells CMSETUP to install the product on a new machine.

The keyword CMSource specifies from where the product files are to be installed. The value for CMSource is most often the directory to which the CMIMAGE tool has loaded each disk on a LAN-accessible hard drive. The source can also be specified on the CMSETUP command with the /S parameter, which overrides any CMSource keyword in the response file.

The keyword CMMModelCFG indi-

*Communications
Manager/2 response
files need not contain
every parameter to be
configured.*

cludes keywords such as Local CP Name and Local Node ID to identify that workstation. When the response file install process is run, it applies the keywords to the model configuration file and generates a user configuration file, as shown in Figure 3. This approach works well for appli-

Now...

UCANDU Visual ProgrammingTM with REXX

"...OS/2's time may be here, thanks to a \$299 tool called Visual Programming with REXX which brought the house down at a recent OS/2 conference in Colorado...it is to REXX and OS/2 what Visual Basic is to Windows and DOS."

—Robert X. Cringely
InfoWorld, January 25, 1993



Now you can unleash the power of the REXX language in a GUI environment with an easy-to-use visual programming tool. UCANDU Visual Programming with REXX offers a CUA '91 interface, WYSIWYG editors, and drag-drop programming—features that allow you to visually construct OS/2 2.0 applications in record time.

Who says powerful results can't be easy to achieve?

Because VP is completely integrated with the OS/2 2.0 Workplace Shell, you can leverage your existing skills. VP offers multiple views, drag-drop interaction, pop-up menus, settings notebooks, direct editing, and other CUA '91 features, and supports the OS/2 2.0 font and color palettes. Even users with little or no knowledge of programming become productive VP users in no time.

Here's what UCANDU with Visual Programming:

- Quickly prototype and develop OS/2 2.0 CUA '91 applications
- Generate a small, single .EXE file for license-free distribution
- Build client-server programs
- Migrate existing REXX procedures to the OS/2 2.0 GUI environment

UCANDU Visual Programming with REXX gives you access to these OS/2 2.0 programming features:

- Buttons, lists, graphics, sliders, and all the CUA '91 controls
- Business graphics
- APPC, HLLAPI, and OS/2 2.0 Database Manager

Best of all, it's small

System requirements:
OS/2 2.0, 5 Mb memory,
2 Mb hard disk space

\$299
Introductory price

OS/2 2.0, CUA, and Workplace Shell
are registered trademarks of
International Business Machines Corporation.

**UCANDUTM
SOFTWARE**

P.O. Box 336
Cary, NC 27512-0336
Sales (919) 387-7391
Tech Support (919) 380-0616
Fax (919) 380-0757
© 1993 Ucandu Software, Inc.



cates the filename of the model configuration, which is the base configuration that the keywords in the response file will customize for use by a particular workstation.

Just as CMModelCFG can be thought

of as specifying the INPUT configuration to the response file process, CMUserCFG specifies the OUTPUT configuration after all response file keywords have been applied. CMUserCFG is used for both input and output

when no model configuration is specified; this is how configuration changes are made locally after installation.

USAGE SCENARIOS

We will now examine a few uses for response files in Communications Manager/2. We'll start with using response files to install and configure a new user of Communications Manager/2.

New Installation. When installing a new workstation, a valid configuration must be created before installation can be completed. This is done by specifying configuration keywords, with or without a model configuration file. Figure 4 gives an example of a response file, with no model configuration provided, that will install a new workstation to use the 3270 Emulator with a Token Ring card for connectivity.

The configuration keywords Local_CP, Logical_Link, LAN_DLC, 3270_Session, 3270_Color, AT_Keyboard, and Enhanced_Keyboard contain other keywords rather than a single value. These keywords are also called records.

Communications Manager/2 is configured by creating, modifying, cloning (copying), and deleting these records using response files. The keywords and values enclosed by the parentheses configure the record identified by that record keyword. This method is appropriate for applications that need to support logically grouped configuration values.

The keywords in each record correspond to the entry fields, check boxes, radio buttons, and other Presentation Manager controls used by the CMSETUP user interface to configure Communications Manager/2. For example, to represent the Presentation Space size field configured with radio buttons, values of 1 through 5 were assigned to keywords representing each radio but-

```
CMUpdateType = 1
CMUserCFG     = TR3270

Local_CP = (
    Name = NETWORK.MKG49442
    CP_Alias = MKG49442
    Host_FP_Support = 1
    Host_FP_Link_Name = HOST0001
    Node_ID = 49442
)

Logical_Link = (
    Name = HOST0001
    DLC_Name = IBMTRNET
    CP_CP_Session_Support = 1
    Destination_Address = 400078825640
    Solicit_SSCP_Session = 1
)

LAN_DLC = (
    Name = 0
    CaSM_LAN_ID = MKG49442
)

3270_Session = (
    Name = A
    Session_Autostart = 1
    AT_Keyboard_Name = ACSCATUS
    Enhanced_Keyboard_Name = ACSCENUS
    Color_Name = STDCOLOR
    NAU_Address = 2
    Host_Link_Name = HOST0001
)

3270_Color = (
    Name = STDCOLOR
)

AT_Keyboard = (
    Name = ACSCATUS
)

Enhanced_Keyboard = (
    Name = ACSCENUS
)
```

Figure 4. New install response file example

ton. When a record is created, defined default values are taken for keywords not specified in the response file. To start defining keywords for your application, start by looking at your user interface. The following approach for handling repeating entries may also be helpful.

The keyword **Name** identifies which instance of the record to configure. In our example, we configure the 3270 session **A**. Multiple 3270_Session records can be used in a response file to configure more than one session; each would use a different value for **NAME** to identify its session.

The use of a model configuration can reduce the size of the response file needed to create the workstation configuration. If CMSETUP were used to create a model configuration for the 3270/Token Ring feature, the response file from Figure 4 could be shortened to the example in Figure 5. The keywords that remain are those that the 3270 Emulator feature requires to identify that particular workstation to the network; they modify the existing records in the model configuration.

Configuration change. Once the workstation is installed and running Communications Manager/2, let's suppose that some requirements have changed:

- The workstation requires three 3270 sessions instead of just one.
- All three sessions are to be 44 rows by 80 columns instead of the default 25 rows by 80 columns.

These changes can be made by the response file in Figure 6. In this example, the value of **CMUpdateType** (3) indicates that configuration changes are being made. If additional files were required due to these changes, the installation would begin as soon as the key-

words are processed and the configuration verified.

The first 3270_Session keyword makes session **A** 44 rows by 80 columns (represented by **space_size** with a value of 3). The second uses the **Copy** keyword to copy the entire **A** session into a new record, named **B**. All configured keywords of ses-

sion **A**, including the change to the 44-by-80 session size, are duplicated into a new record in the configuration. We are not done creating our second session, however. If we simply used the **NAME** and **COPY** keywords, this configuration would fail verification because certain values are required to be unique



CAPTURE YOUR SCREEN

VIEW YOUR CLIPBOARD

From the leader in OS/2 Screen Image Utilities; **PrntScrn™** is an economical combination of four important utilities:

1. Screen image print/capture, 2. Clipboard Viewer,
3. Screen Blanker, 4. Date & Time Display

PrntScrn is the utility of choice:

For developers writing OS/2 PM applications. Our OEM arrangements allow developers to bundle **PrntScrn** with their application. We handle the graphics print routines so they can concentrate on application details.

For authors, trainers, and courseware developers who need to include OS/2 Workplace Shell images in their material.

For Insurance Companies, Banks, Medical Facilities, and other businesses who need to output *quality* Black & White copies of the screen images displayed by their OS/2 PM apps.

PrntScrn is CUA compliant and LAN compatible. All internal operations are done "by the book" to provide the most stable and reliable product available. Entity licensing available. Includes OS/2 2.x (32-bit) and OS/2 1.3 (16-bit) versions.

Available from major software resellers, or contact:

MITNOR Software
28411 E. 55th Street
Broken Arrow, OK 74014
Phone: (918) 357-1628 Fax: (918) 357-2869



```
CMUpdateType = 1
CMModelCFG   = MODL3270
CMUserCFG    = TR3270

Local_CP = (
    Name = NETWORK.MKG49442
    CP_Alias = MKG49442
    Node_ID = 49442
)

Logical_Link = (
    Name = HOST0001
    Destination_Address = 400078825640
)

LAN_DLC = (
    Name = 0
    CaSM_LAN_ID = MKG49442
)
```

Figure 5. New installation with a model config and response file

```
CMUpdateType = 3
CMUserCFG    = TR3270

3270_Session = (
    Name = A
    Space_Size = 3
)

3270_Session = (
    Name = B
    Copy = A
    Long_Session_Name = SESSIONB
    NAU_Address = 3
)

3270_Session = (
    Name = C
    Copy = B
    Long_Session_Name = SESSIONC
    NAU_Address = 4
)
```

Figure 6. Configuration Change response file

```
CMUpdateType = 3
CMUserCFG    = TR3270

3270_Session = (
    Name = C
    Delete
)
```

Figure 7. Configuration deletion response file

across the range of configuration records. For Token Ring 3270_Session records, those values are NAU_Address and Long_Session_Name.

The third 3270_Session keyword is similar in function to the second. Session C is created using the configuration values of session B, then the fields NAU_Address and Long_Session_Name are set to unique values.

To demonstrate how records can be deleted from the configuration, let's say the customer in our example has decided that the third 3270 session is no longer needed. The deletion process is demonstrated in Figure 7.

SUMMARY

We have seen that the response file redirected installation and configuration function is provided in Communications Manager/2, and have learned how applications can query and update data. With this knowledge, a customer can install applications on a large number of OS/2 workstations. The techniques here can be used for remote installation via response file; the developer is encouraged to provide this support.

Bill Halterman, IBM Networking Systems Line of Business, P.O. Box 12195, Research Triangle Park, N.C. 27709. Halterman is a senior associate programmer in the OS/2 Communications Manager/2 Installation Services department at IBM. He is currently the development team leader for the implementation of CID support in Communications Manager/2. He joined IBM in 1989 after earning a B.S. in computer science from Michigan Technological University in Houghton, Mich.

Don Richards, IBM Networking Systems Line of Business, P.O. Box 12195, Research Triangle Park, N.C. 27709. Richards is an advisory programmer working on Communications Manager/2. He started his career working in development and test areas for the Distributed Office Support System (DIS-OSS) product. Richards joined the design department for Communications Manager/2 in 1990 and is currently working in the area of installation, configuration, and CID. He joined IBM in 1981 after earning a B.S. in computer science and business administration from Taylor University in Upland, Indiana.

REFERENCES

IBM Communications Manager/2 Version 1.0 Network Administration and Subsystem Management Guide (IBM Doc. SC31-6168)

OS/2 2.0 Remote Installation and Maintenance (IBM Doc. GG24-3780)

Automated Installation for CID Enabled Extended Services, LAN Server v. 3.0 and Network Transport Services/2 (IBM Doc. GG24-3781)

Network Transport Services/2 Remote Installation and Configuration Guide (IBM Doc. S96F-8488)



PEN SHUTTER *Screen Capture & Conversion For OS/2*

MORE FUNCTION, LESS PRICE....WHAT A CONCEPT!

At One Up Corporation, abundance in function without abundance in price is paramount. Consequently, you'll find our screen capture application, Open Shutter, provides the flexibility you need to capture, modify, and output your desktop images at a price you'll want to photograph!

CAPTURE YOUR IMAGES

Our easy-to-use capture process allows you to select any rectangular area, window, or the entire desktop and capture with a single user-defined keystroke or mouse click.

MODIFICATIONS MADE SIMPLE

Rotate your image at any angle, map your colors and modify your color palette, and stretch or compress your image to any dimension you need. Preview / compare multiple modified versions of your captured image prior to output.

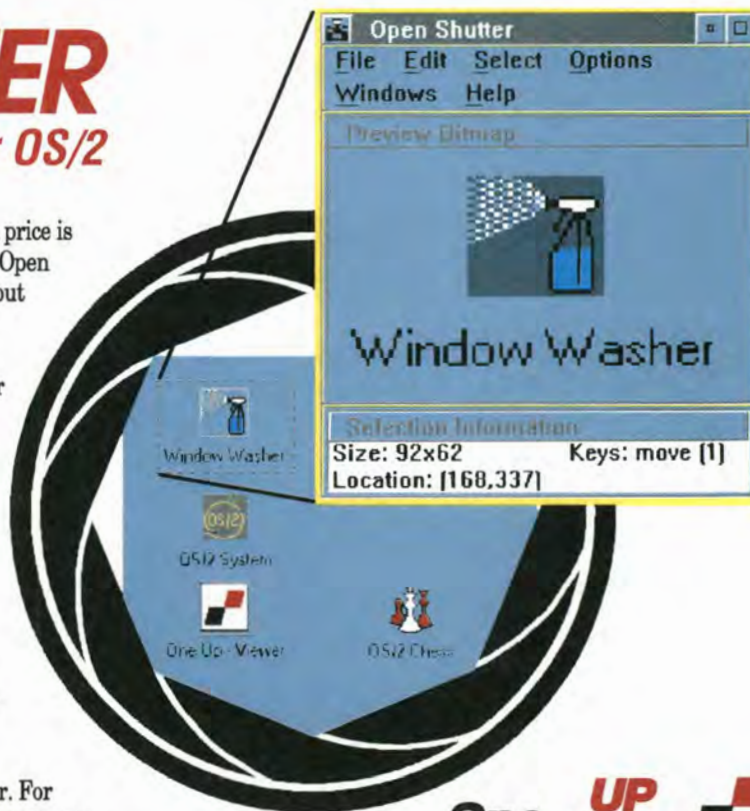
WHAT ABOUT OUTPUT?

Open Shutter offers a varied selection of output devices including printer and clipboard. For soft copy, save your image as an OS/2 or Windows BMP, ICO, or metafile. Also, save as an OS/2 pointer, a Windows cursor, or PCX, TIFF, GIF, PICT formats and more!

So don't get discouraged by high price applications that don't deliver. For only \$59.95, Open Shutter offers you a cost conscious solution for your screen capture requirements. Ask us about our competitive upgrade price, too.

OS/2 is a registered trademark of IBM Corporation

Windows is a registered trademark of Microsoft Corporation



One UP Corporation
1603 LBJ FREEWAY, SUITE 860
DALLAS, TEXAS 75234
1-800-678-01UP

Worldwide Developer
Assistance Program

It

It's fast relief from those annoying technical nits that can drive you buggy. It's answers to questions. Solutions to problems. It's help when you need it.



From development through generating market demand for your OS/2® software products, the Worldwide Developer Assistance Program (DAP) is there for you.

It's technical support and a database of common OS/2 questions and answers available through CompuServe® and other worldwide online systems. It's *IBM OS/2 Developer*, a magazine featuring technical and "how-to" articles on exploiting the rich features of OS/2. It's discounts on IBM PS/2®

even

kills

hardware, software and publications. It's an electronic repository of OS/2 software development tools and demos available at no cost. And it's more.

Early Code Programs let you test new OS/2 versions with your product. Migration Workshops provide software and technical assistance to port applications to OS/2. But the Developer Assistance Program doesn't end with the completion of your software program.

Membership entitles you to free listings on our OS/2 CD-ROM Application Demonstration System and application directory. Take advantage of direct marketing offerings at reduced prices through the OS/2 Direct Marketing Center. And

there are many more offerings, including business shows, targeted mailings, in-package promotions and instructional videos.

To find out more about the DAP and how to qualify for specific programs, enter "GO OS2DAP" on CompuServe, or call 1 407 982-6408. You'll see we've developed a powerful program to help develop yours.

bugs.



Our CD-ROM Professional Developers' Kit provides beta-level code for operating system and tools and includes online technical reference libraries.





This article describes a communications client/server product that enables resource sharing on a server workstation across a group of client workstations. **BY JULIE KING and CHARLES GREEN**

Extending Application Interfaces For Communications: Client/Server

THE CLIENT/SERVER MODEL FOR processing has become popular in the computing industry. There are many varieties of this model, including database and application client/server. This article describes a communications client/server product that enables resource sharing on a server workstation across a group of client workstations. This provides connectivity to Enterprise Systems Architecture/390™, Application System/400™, RISC/6000™, OS/2, DOS, or Windows™ workstations.

IBM Communications Manager Client Server/2 was designed to allow sharing of server-based system and communications resources, such as links, adapters, and communications code by a client workstation. By enabling LAN-attached client workstations to share server resources, the client's RAM requirements are dramatically reduced over a full implementation while still offering the user a rich set of system network architecture (SNA) communications programming interfaces. Additionally, DOS and Microsoft Windows 3.x client workstations can participate fully in the SNA networking environment.

INTRODUCTION AND OVERVIEW

The IBM Communications Manager Client Server/2, when installed as a communication server on an OS/2 workstation, provides a set of communications protocols that can be shared with DOS, Microsoft Windows 3.x, or OS/2 client workstations using 286, 386, or

486 processors. (The OS/2 clients will be supported in May 1993, while DOS and Windows support will be added by the end of the year. Descriptions of this support are accurate as of press time; they are, however, subject to change before May.)

The communication clients may be attached to the server on a LAN by a Token Ring, PC network, or Ethernet. From the server, the full set of connectivities offered by Communications Manager/2 are available to connect to an SNA network, including Synchronous Data Link Control (SDLC), X.25, Integrated Services Digital Network (ISDN) and LANs, as shown in Figure 1.

The general structure of the communications client support provides a set of APIs at the client workstations supported remotely at the server. The APIs provided are advanced



Julie King



Charles Green

The APIs are designed to be as close to the existing Communications Manager APIs as possible to simplify the porting of existing applications.

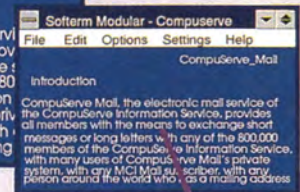
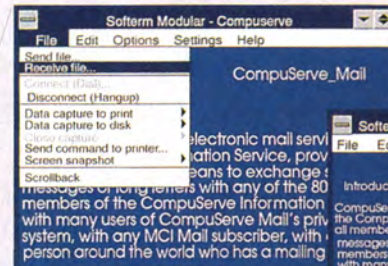
program-to-program communications (APPC) for basic and mapped conversations, common programming interface-communications (CPI-

NEW! SOFTTERM[®]++ FOR OS/2[®]

Only \$50

The replacement for Softterm[®] Custom that is bundled with your OS/2[®]

Concurrent Sessions



New!

Graphic Terminals
DEC 340
DEC 240, 241
Tektronix 4010, 4014

New!

Dynamic Font
132 Column Support

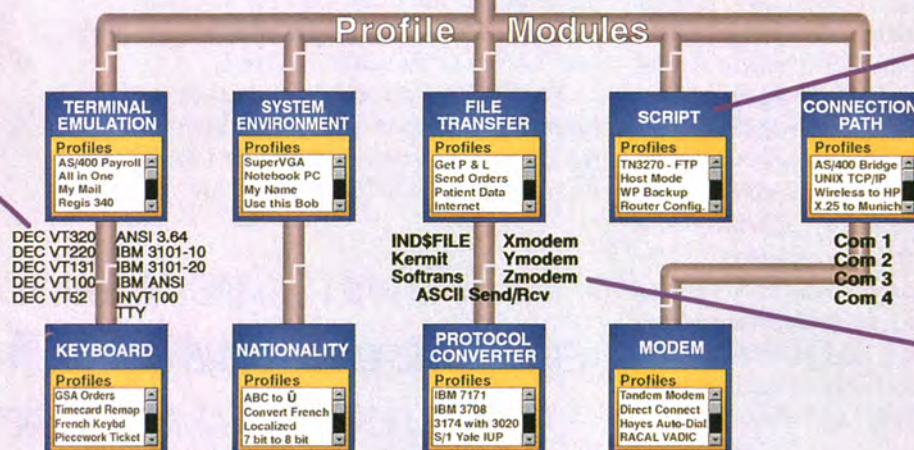
Procognitive[™]
The Softterm Graphical
User Interface

Also New!

DEC 320,220

New!

Scripts
Genome[™], The
Softterm Script Language



New!

Zmodem
IND\$FILE

"The only high-performance graphic terminal comm program for OS/2[®]."

800-225-8590

Softronic, Inc. 5085 List Drive, Colorado Springs, CO 80919 FAX: 719-548-1878

OS/2 is a registered trademark of IBM Corp.

C), conventional LU applications programming (LUA), plus a subset of the API provided by Communications Manager for some common services. When an API is called, the data required by the call is packaged and transported over a NetBIOS connection to get to the server. At the server, the data is received from NetBIOS, and the SNA communication code is executed. When processing completes, the returned data is again transported across the NetBIOS connection back to the client workstation.

The communications partner may reside anywhere in the network. An application on one client can communicate with an application on another client, on a host, on another workstation, or on the server. The remote API is shown in Figure 2.

PROGRAMMING INTERFACES

The APIs offered provide SNA LU types 0, 1, 2, 3, and 6.2 protocols, along with the ability to transfer network management data, convert between EBCDIC and ASCII, and get code page tables. The Communications Manager/2 API support disk provided with the Communications Manager/2 application contains sample programs written in C, COBOL, and macro assembler (MASM) language, and libraries that support programming these languages.

The APIs are designed to be as close to the existing Communications Manager APIs as possible to simplify the porting of existing Communications Manager applications. A few API changes occurred when the underlying operating system support made support of the exact interface impossible or when minor additions to the interface made the management of clients much easier.

For example, in APPC, all verbs except for `RECEIVE_AND_POST` are supported exactly as in Communica-

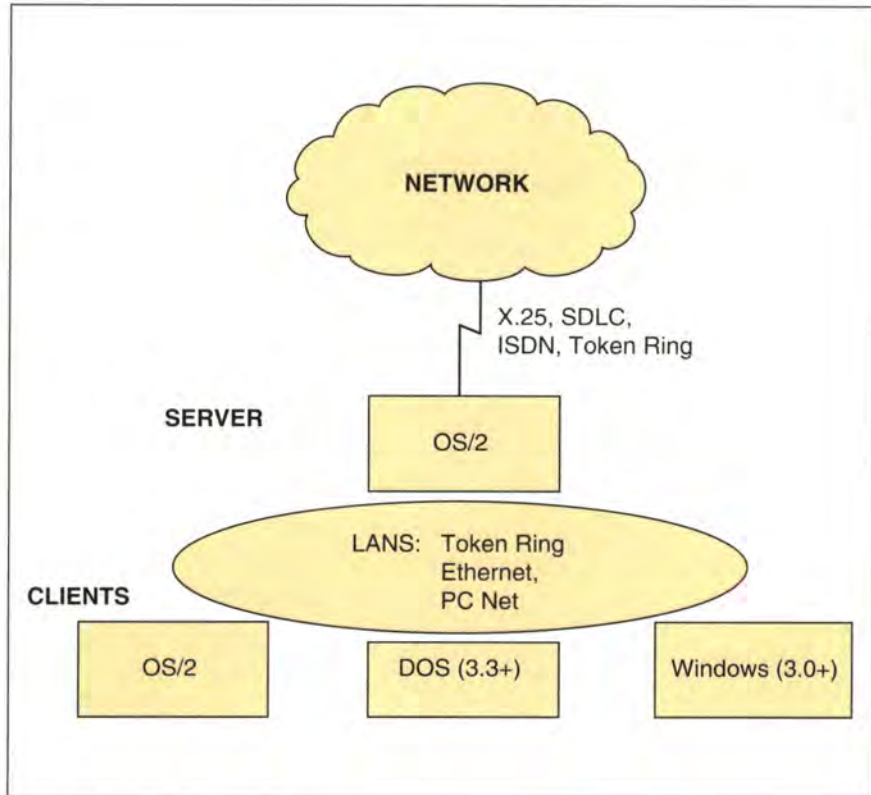


Figure 1. Network connectivity

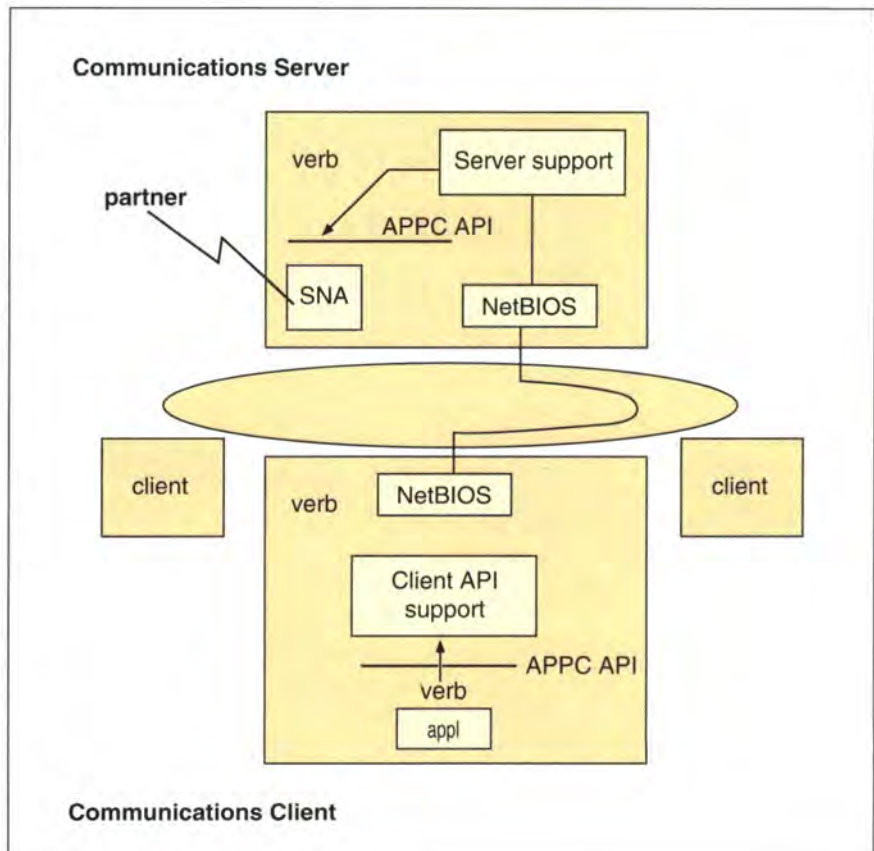


Figure 2. Remote API



tions Manager/2. Since the RECEIVE_AND_POST verb in OS/2 uses a semaphore to receive notification that the verb has completed, its support on the OS/2 client remains the same, while a window handle mechanism was required for the Windows environment. The RECEIVE_AND_POST verb is not supported on DOS clients.

Also provided with the communications client product is a 3270 and 5250 emulator that makes use of the remote LUA API. The client emulator also makes available the EHLLAPI and SRPI interfaces.

TECHNICAL CHALLENGES

Some of the key technical challenges were to provide the communications client with easy and continuous access to server resources while

providing the network administrator with configuration ease of use, control, and the flexibility to distribute available resources.

Finding a Server. One of the goals of the communications client/server design was to keep the client easy to install and configure. The client should be able to find a server easily, with no system definition overhead.

To avoid any need to define a server name to the client workstations, the communications client uses a NetBIOS datagram to broadcast its need for a server. Available servers will return their name and the class of services they are able to

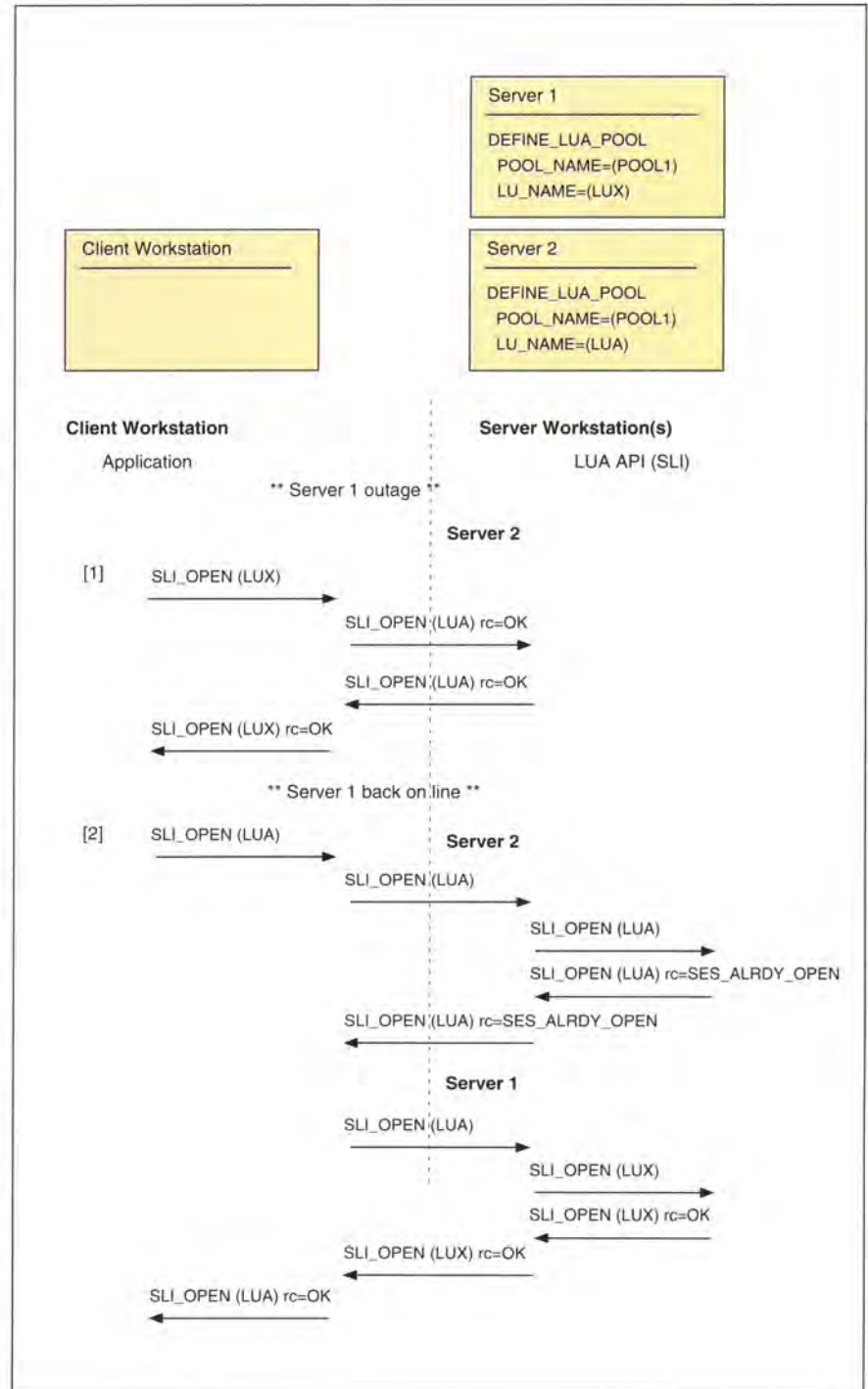


Figure 3. LUA pools

provide. For example, a server may respond that it can provide APPC, network management, and CPI-C or LUA services. The client then establishes a NetBIOS session with the server and receives a list of resources.

By offering granularity in the types of services provided by the server, a network administrator has some flexibility in setting up the network. For example, if one workstation has a link to the host, it can be configured as the server for 3270

Sometimes ease-of-use and availability conflict with a network administrator's need to control and operate the network.

FIND, SEE & MANAGE OS/2 FILES IN A FLASH!

***Introducing
The Norton Commander®
for OS/2:***

From Peter Norton, the master of utilities, here's the first OS/2 graphical file management utility for your PCs and your network.

***See Your Files Without
Opening Applications.***

You can quickly search your entire system for files or folders. They'll be listed in a display that lets you view any file's contents and launch or edit it instantly.



Customize your menus to launch applications instantly.



***Powerful
Application
Menus.***

You can create standardized read-only menus for network users. Users can easily customize their own menus for local applications, too.

***Low Cost,
High Efficiency.***

Personal and Network Directories Revealed.

In a single window, you can view copy, move, or delete files or change drives. Compare folder contents with differing files highlighted. Drag and drop folders between file display panels or directly to the desktop.

**To order,
contact your local dealer
or call**

1-800-628-4777
ext. AD80

800 # valid only in USA

These indispensable utilities will save you countless hours of frustrating searches, from the day you install them. Don't be surprised if The Norton Commander pays for itself within a week!





UNIX[®] POWER AT OS/2[®] PRICES

HIPPIX[™] defines a set of command-line utilities and programming libraries that give the OS/2 2.0[®] user the power and features of the UNIX operating system, without the cost of installing and maintaining UNIX.

HIPPIX commands comprise over 100 utilities implementing most of the IEEE POSIX 1003.2/1003.2a draft standards including awk, grep, more, sed, sh, and vi plus programming utilities like lex, make, the rcs revision control system, and yacc.

HIPPIX programming libraries provide more than 150 functions, supporting over 90% of the POSIX 1003.1 system API. Applications you develop are source code compatible with POSIX compliant UNIX systems. Developers may freely distribute the HIPPIX DLL along with their applications.

HIPPIX commands and programming libraries for OS/2 2.0 are available for a limited time for \$179. The commands are available separately for \$129. For further information, contact Hippo Software at 723 60, 2675 on CompuServe or from the Internet use 72360.2675@compuserve.com.

HIPPIX may be ordered from Pacific HiTech, Inc. at (800) 765-8369.

Hippo Software, Inc.

448 East 400 South, Suite 303, Salt Lake City, UT 84112
(801) 531-1004 ♦ (801) 531-1302 FAX

UNIX is a registered trademark of Unix Systems Laboratories.

Code: LBS1

LOOKING FOR FAST ANSWERS TO YOUR DEVELOPMENT QUESTIONS?

You need to do two things:
Go on-line with CompuServe
Information Service. Use
Golden CommPass to do it.

CompuServe hosts OS/2 developer and user forums monitored by technical personnel. IBM developers are there with responses to anything

that's got you stuck. Other OS/2 programmers and enthusiasts are there, too, comparing notes and giving feedback. It's definitely the place to be.

Time is money when you connect to CompuServe, and Golden CommPass saves you both. It lets you compose your questions off-line, send them in a flash and disconnect. It gets your replies while you're busy programming. It's a native OS/2 app, with all the power and features you'd expect.

Get the fastest access to answers. Golden CommPass keeps your development schedule on course. The fact is, the sooner you start using our program, the sooner they'll be talking about *yours!*



Golden CommPass[™]

CompuServe[®] Access Software

Creative Systems Programming Corporation
(609) 234-1500 • Fax: (609) 234-1920 • CompuServe: 71511,151

NDP Fortran, C/C++ and Pascal Compilers

- VMS, VS and MS extensions
- Compatible with Framework and C Set — Fortran can call C Set or NDP C
- Uses IBM's Link386 to generate native OS/2 executables
- Can directly call the OS/2 APIs
- Generate globally optimized 32-bit mainframe quality code
- Support for all coprocessors and the i860
- Available for DOS, OS/2, UNIX, NT and Coherent
- 3rd party numerics and GUI Class libraries available

For more information, call our Tech Support Group
at (508) 746-7341.

Microway[®]

Research Park
P.O. Box 79
Kingston, MA 02364

Document Imaging Software Development

Sigma Imaging Systems is looking for experienced PC Software Engineers to work on the development of client-server document imaging systems using MS-Windows and OS/2. You must have a BSCE or BSEE and at least 4 years of professional C/C++ development experience. Proficiency with Windows or OS/2 PM is highly desirable as well as strength in one or more of the following areas:

- Database Management
- GUI Programming
- LAN/WAN Applications
- Operating Systems Internals

Knowledge of object oriented design and programming is a plus.

We offer an interesting and challenging work environment, competitive salaries, excellent benefits, and an employee stock option plan. Please send resume to Sigma Imaging Systems, Inc. 622 Third Avenue, 30th Floor, New York, NY 10017. Equal opportunity employer.

SIGMA

IMAGING SYSTEMS, INC.

emulator sessions. At the same time, it does not have to be the server for CPI-C applications. It is possible to share the server workload, by splitting up the job assignments across multiple servers.

When a client application starts, the client will select a server, choose a resource, and send the API request over to the appropriate server. The client uses a NetBIOS session for each connection with a server, with the number of NetBIOS sessions available to the client configurable. If the default value of four is used, then there can be a maximum of four simultaneous servers.

Hot Backup and Load Balancing.

Another goal was to maximize availability, by using multiple servers to provide backup capability. Because of differences between SNA LU types that are dependent on the host for support and independent LU 6.2, different design approaches were taken to provide backup capability.

In the LU 6.2 scenario for an APPC or CPI-C application, one server was designated as the preferred server when defining access for a list of client machines and users. This provides the network administrator with the capability to distribute users to available capacity. At one or more other servers, access can be defined for those client machines and users without the preferred server designation. When the client finds the preferred server, the client will use it. If the preferred server is not available, the client will find a backup by using one of the others to which it has access.

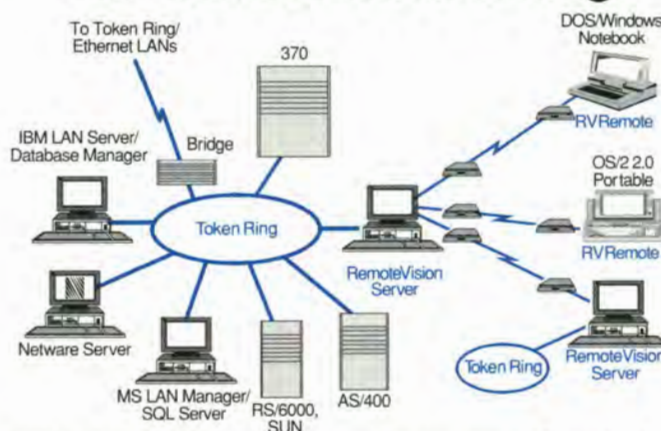
An LU will be dynamically defined at the server for the client's use as a backup server. If a backup server is used and a conversation is in progress when the preferred server becomes available, at transaction program completion the

client will switch to the preferred server. The LU it was using at the backup server will be dynamically deleted, and one at the preferred server will be defined. The user will be unaware the client has switched servers.

For LUA applications, which use

LU0,1,2,3 protocols to enable load balancing and backup server capability, the LU pool concept is used. An LUA pool can be defined at one server or defined to span multiple servers. When the pool spans multiple servers, clients will select another server who shares the same

Introducing RemoteVision.[™] The Complete Software Solution For Remote Networking.



- Extend LANs transparently over asynchronous modems to remote workstations and workgroups.
- Run DOS, Windows[™], & OS/2[™] apps written to multiple protocol stacks on remote nodes.
- Operate remote computers as full-function LAN nodes.
- Connect to remote offices with LAN-to-LAN dial-up networking.
- Add remote connectivity services using existing LAN machines.
- Access remote PC servers and hosts without additional host gateways.
- No specialized hardware/adapters.

RemoteVision makes it easy and affordable for DOS, Windows and OS/2 users in remote locations to connect to Token Ring or other LANs using dial-up modems. Response time remains fast, applications don't need changing, and users won't need retraining. So you can turn your remote machines into full-function workstations. With RemoteVision. To immediately find out more, or to take advantage of our "Try Now, Pay Later" 30-day money-back guaranteed trial offer, call today.



1265 Montecito Ave., Ste. 101, Mountain View, CA 94043
Tel: (415) 965-8607, Fax info: (415) 965-8607, (then press 2)
Trademarks are from their respective companies. ©1993 Token Technology, Inc. 001-0

pool if one server is down. If all servers are operating, the load will be randomly spread across those servers willing to share their resources. Figure 3 shows an example of pool configurations that maximize backup capabilities.

In this example, a pool named `POOL1` is defined to span two servers by creating an LUA pool definition at both servers. At a later time Server 1 becomes unavailable due to an outage. If a client application issues an `SLI_OPEN` (or `RUI_INIT`) to a LU name on the unavailable server, the communications client will use one of the LU names out of `POOL1` for Server 2 so that session establishment can still proceed. Now, assume Server 1 becomes available again. An `SLI_OPEN` issued for a busy LU name in `POOL1` on Server 2 will be able to use an LU on Server 1 instead.

Access Control. Sometimes ease of use and availability conflict with a network administrator's need to control and operate the network. When a client is requesting a server, all servers will respond unless configured differently. The client can find a server across a bridge and may choose a server that is not the closest in proximity. While this is desirable for availability, the network administrator may choose to assign certain machines to one server or perhaps keep clients away from a particular server. The ability to control access comes through creating access definitions at the server.

Access definitions, created based on the type of resource being protected, indicate who is allowed to use them. The definitions allow the use of pattern-matching symbols—wildcards—to make them easier to create. For instance, in Figure 4, when setting up an LUA pool to be used by the emulator, you could define a pool that any machine name and user name could use, or a definition could be very restric-

```

DEFINE_LOCAL_CP  FQ_CP_NAME(APPN.EMULATOR  )
                  CP_ALIAS(EMULATOR)
                  NAU_ADDRESS(INDEPENDENT_LU)
                  NODE_TYPE(NN)
                  NODE_ID(X'05DC0001')
                  HOST_FP_SUPPORT(YES)
                  HOST_FP_LINK_NAME(HOSTLINK);

DEFINE_LOGICAL_LINK  LINK_NAME(HOSTLINK)
                     ADJACENT_NODE_TYPE(LEARN)
                     DLC_NAME(IBMTRNET)
                     .
                     .
                     .
                     ;

DEFINE_LUA  LU_NAME(LUA1  )
            HOST_LINK_NAME(HOSTLINK)
            NAU_ADDRESS(1);

DEFINE_LUA  LU_NAME(LUA2  )
            HOST_LINK_NAME(HOSTLINK)
            NAU_ADDRESS(2);

.
.
.

DEFINE_LUA  LU_NAME(LUA100)
            HOST_LINK_NAME(HOSTLINK)
            NAU_ADDRESS(100);

DEFINE_LUA  LU_NAME(MYLUA)
            HOST_LINK_NAME(HOSTLINK)
            NAU_ADDRESS(101);

DEFINE_CLIENT_LUA_ACCESS  LUA_ACC_NAME(LUASVR1)
                          MACHINE_NAME(CLNT*  )
                          USER_OR_GROUP_ID(*)
                          LU_NAME(*)
                          POOL_NAME(LUAPOOL);

DEFINE_CLIENT_LUA_ACCESS  LUA_ACC_NAME(LUASVR2)
                          MACHINE_NAME(MYTERM  )
                          USER_OR_GROUP_ID(USER)
                          LU_NAME(MYLUA);

DEFINE_LUA_POOL  POOL_NAME(LUAPOOL)
                  LU_NAME(LUA*  );

```

Figure 4. Access Definitions using wildcards

tive—only this user on this workstation can use this LU resource.

If LAN Services is installed on the communications client and server, it can be called to obtain the machine names and user names used to control access. If they are not installed, then user-provided names are used. In either case, the server uses these names to verify access to its resources and will inform clients only about resources to which it has access.

SERVER GATEWAY

Communications Manager/2 provides the SNA Gateway as "a non-dedicated server PC that provides its clients a shared link to an IBM SYSTEM/370 host. Each client workstation thinks it is directly connected to the host, while the host thinks it is communicating with a single terminal controller."

(Orfali and Harkey, 1992).

The Communications Manager Client Server/2 provides similar function for lower cost to the OS/2, DOS, and Windows client workstations. This feature provides a server gateway that allows communica-

LUA resources.

Tradeoffs exist when using either a SNA or a server gateway. With both options available, a network administrator can tailor the network to match its resources. By providing SNA and server gate-

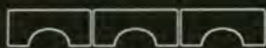


"Each client workstation thinks it is directly connected to the host, while the host thinks it is communicating with a single terminal controller."

tion clients to access server resources without the need to define links and administer network addresses at the client. When using LUA pools, the network address is not hard-coded but selected from a list of available

ways, a workgroup can migrate to a communications client/server technology, choose the best connectivity provided by Communications Manager/2, or choose the best-of-breed emulator.

ATF



Softbridge, Inc.

Some things are best done the old-fashioned way...

Software testing isn't one of them.

Old-fashioned manual methods can't stand up to the challenges of testing applications built for a client/server, graphical user interface environment.

The Softbridge Automated Test Facility products — **ATF NetWorked** and **ATF WorkStation** — can meet those challenges head on. Whether you're developing stand-alone or distributed OS/2 or Windows apps, or building client/server solutions combining both platforms, you should be testing with ATF.

Softbridge, Inc.
125 CambridgePark Dr.
Cambridge, MA 02140
617-576-2257 (Phone)
617-864-7747 (FAX)

Quite possibly, the best software for testing client/server, OS/2 apps.

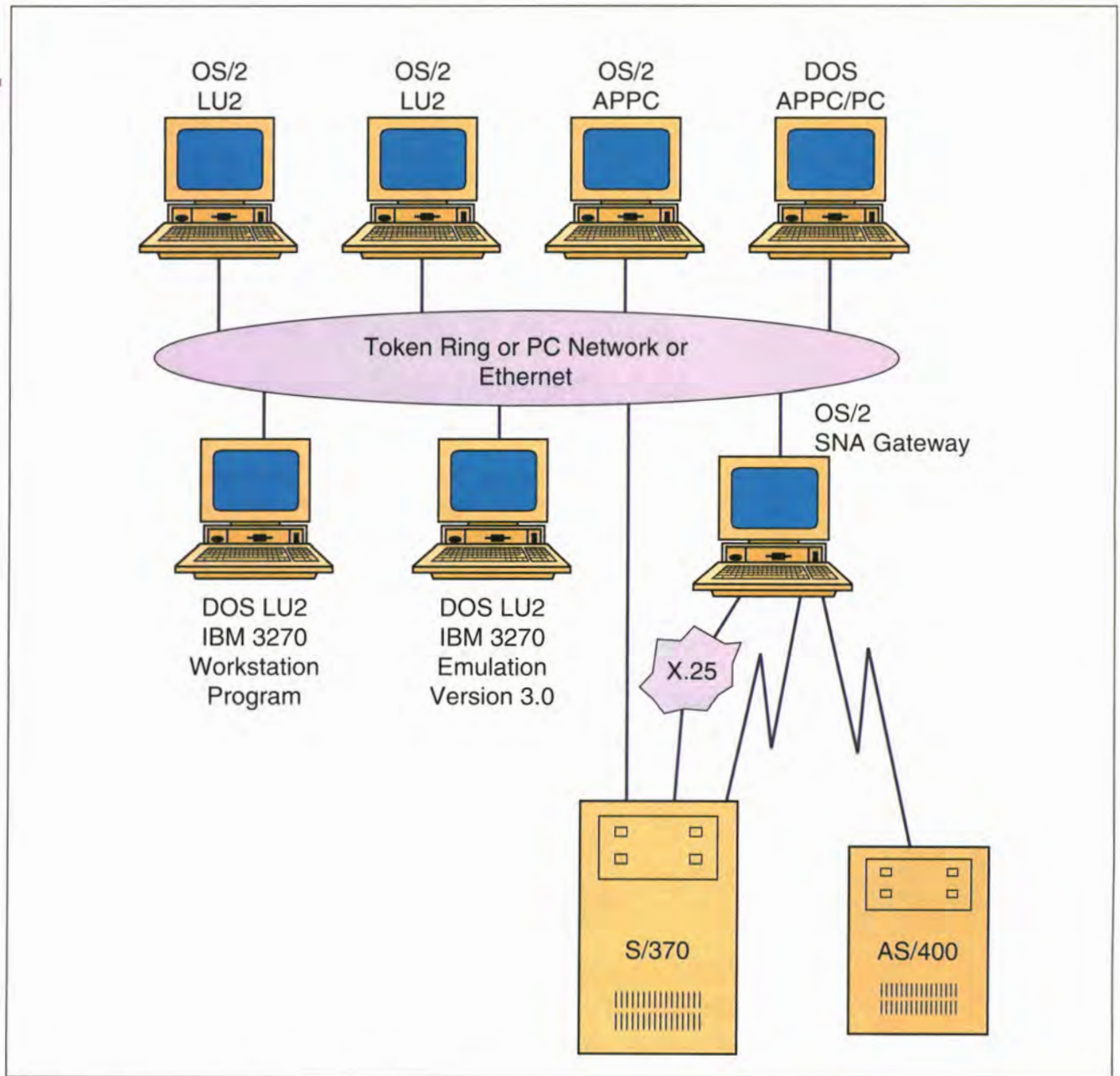


Figure 5. *The Communications Manager/2 SNA gateway*

NETWORK MANAGEMENT

IBM Communications Manager Client Server/2 enables network management features available in Communications Manager/2 at the client workstation, such as error detection, logging, and forwarding of network alerts through the server to an IBM host running NetView. Additionally, alerts may

be routed to IBM LAN Network Manager for logging or filtering before storing or routing to NetView. The client/server product supports user alerts in the form of `TRANSFER_MS_DATA` calls, which enables DOS and Windows clients to fully participate in the customer's network management strategy.

SUMMARY

Communications Client Server/2 will extend a common set of application interfaces across OS/2, DOS, and Windows. This enables the application developer to reuse communications code across those platforms, improving productivity.

Julie H. King, IBM Corp., P.O. Box 12195, Research Triangle Park, Raleigh, N.C. 27709. King is a Senior Programmer in Workgroup Communications Products. She joined IBM in 1980 and has been involved with networking products for 12 years. King has a B.S. in chemistry from Duke University and a M.S. in computer science from the University of Oregon.

Charles E. Green, IBM Corp., P.O. Box 12195, Research Triangle Park, Raleigh, N.C. 27709. Green is an advisory programmer in Workgroup Communications Products. He joined IBM in 1982 and has been involved with networking products for eight years. Green has a B.A. in political science from the University of North Carolina at Chapel Hill.

REFERENCES

For more information about programming applications for IBM Communications Manager Client Server/2, refer to the following publications:

Orfali, Robert and Dan Harkey, *Client/Server Programming with OS/2 2.0, 2d ed.* New York: Van Nostrand Reinhold, 1992.

Communications Manager/2 Application Programming Guide (IBM Doc. SC31-7012)

Communications Manager/2 APPC Programming Guide and Reference (IBM Doc. SC31-6160)

Communications Manager/2 Conventional LU Application Programming Reference (IBM Doc. SC31-6166)

Communications Manager/2 Network Administration and Subsystem Management Guide (IBM Doc. SC31-6168)

SAA CPI Communications Reference (IBM Doc. SC26-4399)



Find Code *Fast* • Change Code *Fast* Learn Code *Fast* • Navigate Code *Fast*

SourceLink™

SourceLink is an OS/2 programming development tool combining the speed and power of hyper-link source code access with a fully functional editor. The edit screens are context sensitive. Click to access on-line help. Click-click to access your source code. With over 100 menu items to choose from, SourceLink is a feature rich programming environment.

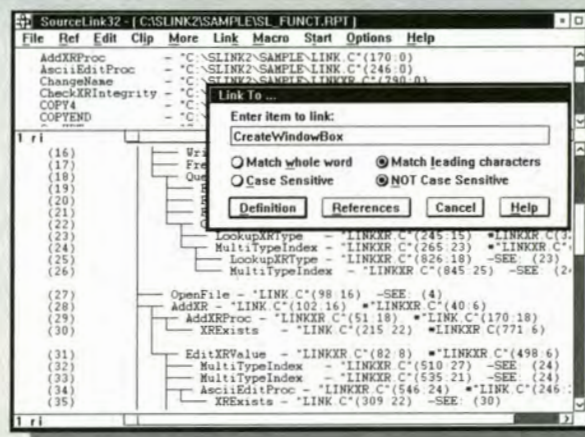
Double click on a function, symbol or other string. Let SourceLink pop up a list of occurrences. Double click on each occurrence and SourceLink will immediately display the source code. Use the integrated editor to change or create new code.

Unmatched in speed and convenience for finding and changing source code. Ideal for porting code, changing code, analyzing code, and creating programs from existing code, templates and samples.

SourceLink will save you valuable programming time and energy! SourceLink is a refreshing new experience in programming!

Features:

- HyperLink source code access.
- Fully functional editor.
- Supports 'C', C++, MASM.
- REXX macro language interface.
- Presentation Manager GUI.
- WorkFrame/2 compatible.
- Generates function call tree displays.
- Context sensitive edit windows.
- Many file manipulation utilities.
- Find Files, Delete, Copy Files/ Directories.
- Context sensitive View Help.
- Multi-File/Dir. string search.
- Project configurable.
- Spawn compiles and commands.
- Create your own code templates.
- Multiple windows.
- Cross reference globals, symbols, dialogs, API calls, and more.
- Also available for OS/2 1.3.
- An OS/2 2.0 32 bit application.



SourceLine Software, Inc. - 7770 Regents Rd #113-502 - San Diego, CA 92122 - (619) 587-4713

Your Satisfaction is Guaranteed

Create GUI Applications That Deliver The Promise Of OS/2.



YOUR WAIT IS OVER!

Now there's a graphical development and decision support system that combines unparalleled data access and reporting with the stability and multitasking power of OS/2 2.0. It's called PM/FOCUS from Information Builders, a leader in application development tools for almost every environment.

EXPERIENCE OUR EASE AND SPEED

PM/FOCUS offers a rich graphical toolset for fast, easy application development with a minimum of coding. All application components, including databases, procedures and forms become simple graphical objects that can be

INTRODUCING PM/FOCUS

controlled by mouse-driven "drag and drop."

Built-in list boxes, check boxes, radio buttons, type fonts and other graphical tools allow you to create attractive GUIs that are so intuitive, they're a snap for even the most unsophisticated end users.

POWERFUL REPORTING FEATURES

PM/FOCUS offers the most powerful reporting language of any product on the market today. And building reports is a breeze. You simply

transform database fields, record selections, and report headings and footers into selectable objects. Even complex sorts are available at the touch of your mouse.

DELIVER GREAT OS/2 GUIs

Get the promise of OS/2 now. Make PM/FOCUS your corporate standard for sensational GUI application development. For more information, or to attend a free seminar...

Call 800-969-INFO

In Canada call 416-364-2760

IBI PM/FOCUS
Information Builders, Inc.



This article discusses improving remote access design using IBM's Distributed Console Access Facility program. The techniques should also apply to most programs that provide remote access capability to OS/2 Presentation Manager (PM) applications. BY GREG LOTEN

Application Design For Remote Access

REMOTE MONITORING AND CONTROLLING of displays on a PS/2 is a fast-growing requirement for users with networks. The Presentation Manager desktop can require over 1MB of video memory to display; to faithfully portray the activity of a remote PS/2 can require large amounts of data to be sent over the network. Because the quantity of data and the speed of transmission affect how quickly changes on the remote PS/2 can be revealed on the monitoring PS/2, performance is a major concern for users (and therefore for application developers). While this article discusses improving remote access design using IBM's Distributed Console Access Facility™ (DCAF) program, the techniques should apply to most programs that provide remote access capability to OS/2 Presentation Manager (PM) applications.

WHAT IS THE DISTRIBUTED CONSOLE ACCESS FACILITY?

DCAF is a remote access program that allows interaction with networked PCs. Version 1.0, available since early 1991, allows users to access the text applications running on DOS or OS/2 1.2 or above. DCAF 1.1 extends this capability to the PM screen group for OS/2 2.x.

DCAF 1.1 can display the screens of several other workstations, each in its own window. All changes on a remote PS/2 screen are reflected in the viewing window; DCAF also allows interaction with a remote work-

station from the observer's keyboard and mouse. Other DCAF features include file transfer capabilities in either direction, the ability to reboot the remote workstation, and security features.

TYPICAL USES

A common use for DCAF is as a centralized help desk. In this setup, a user with a problem telephones the help desk expert, who asks the user for identification and establishes a DCAF session in monitoring mode. The user can then demonstrate the problem while the expert watches. Without leaving his or her desk, the expert can watch the problem



Greg Loten

DCAF 1.1 can display the screens of several other workstations, each in its own window.

occur and catch any user mistakes, then switch to controlling mode and show the user how to correct or avoid the problem.

DCAF can also be used for problem determination at an unattended workstation. A troubleshooter can check if a product is installed correctly or ensure that configuration files are accurate. If they are not, they can be adjusted, or

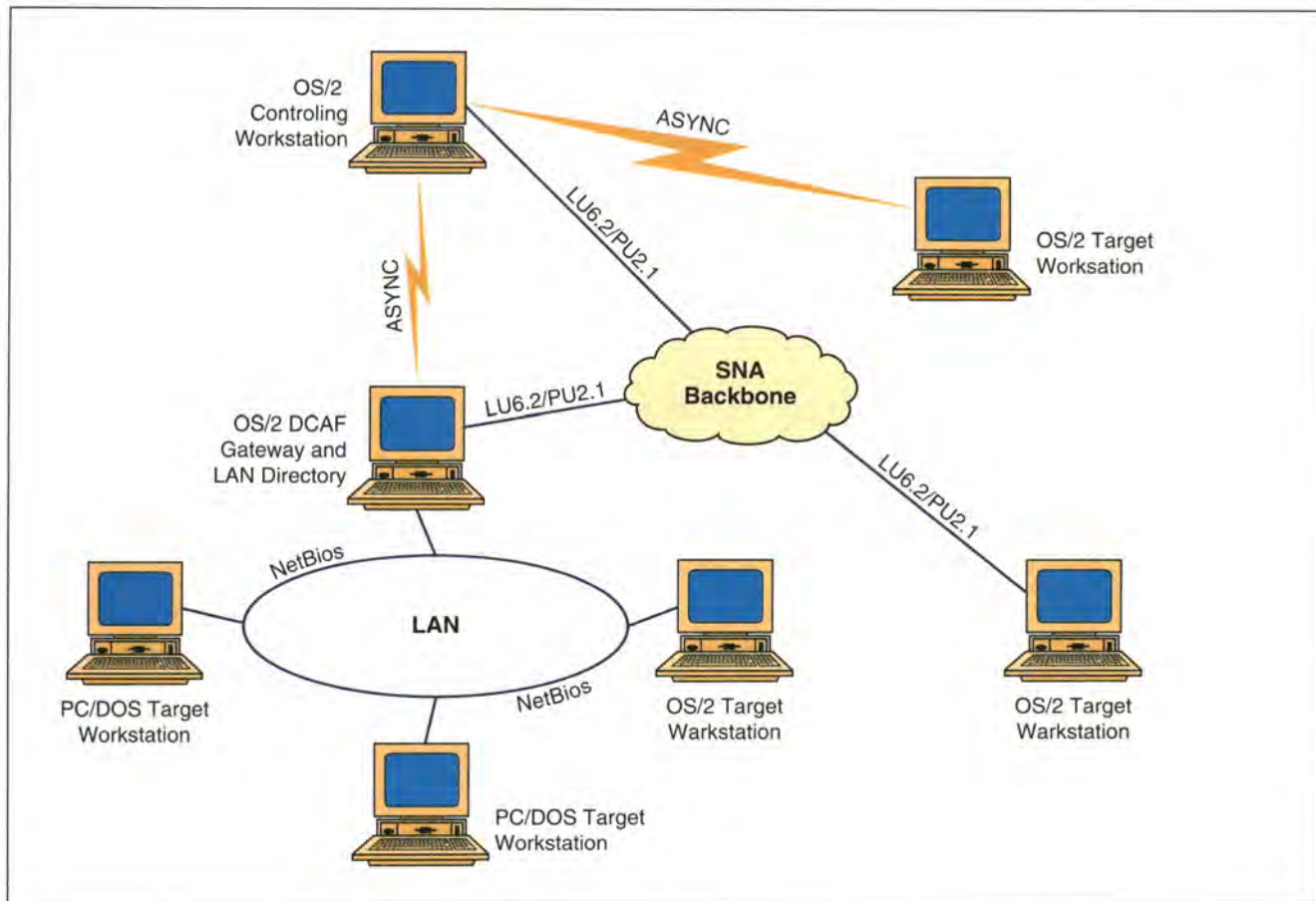


Figure 1. Possible connectivity situations

updated versions can be transferred and the workstation rebooted remotely. DCAF can be restarted

DCAF can have many other uses, from file server administration to remote control of PS/2s in hazardous environments. Any full-screen text or PM application can be operated as if the remote operator was actually sitting in front of the workstation.

CONNECTIVITY VIA APPC, ACPI, OR NETBIOS

Figure 1 shows the connectivity situations DCAF supports. In DCAF parlance, a controlling workstation views and optionally controls one or more target workstations. A DCAF gateway workstation provides connections to targets across a LAN using NetBIOS (the targets are listed by the DCAF LAN directory). A single workstation can function simultaneously as a DCAF gateway, LAN directory, target, and controlling workstation.

PERFORMANCE IS THE KEY

If an application needs to be accessed remotely, consider this during design and implementation. Network management applications are prime candidates, as is any application for which centralized help desk support is needed. Users may test competitive applications to see which ones perform better when accessed remotely.

Typically, applications draw on the screen too frequently and often send unnecessary information, a situation that has little impact on standalone local applications. But on a remote access system, this situation can cost time and bandwidth, or money on packet switched systems. In some of the worst examples, a series of dialogue boxes or windows pop up one after another (often an indication that the program originated on a mainframe).

Applications frequently refresh unchanged areas of the screen. On a remote access system, this can cost time and money.

automatically after the reboot command; the troubleshooter can then verify that the adjustments have worked.

Avoiding these problems and making a program run more effectively when accessed remotely takes an understanding of how remoting applications work.

HOW DOES DCAF WORK?

Most remote-access software products use one of two approaches to reproduce a remote workstation's screen. One way is to allow drawing operations to occur on the target screen and then, on some trigger condition, send the current state of the screen to the monitoring workstation. There are variations of this scheme; the better programs send only changes and compress the data. We will call this the "bitblt" (pronounced "bit-blit") method, after the GpiBitBlt (BIT-wise BLock Transfer) function, as they both operate at the pixel level.

A second approach is to intercept drawing calls to the display driver and send the orders to the monitoring workstation where they are

replayed. Typically this requires a replacement or dummy display driver to fool the operating system into believing it is talking to the original display driver. We will call this the "hooks" method, as we metaphorically hook out the calls to the display driver. The 32-bit graphics engine shipped with OS/2 version 2 has facilities to support both DCAF methods simultaneously.

BITBLT ASSISTANCE

To assist with the bitblt method, OS/2 version 2 has defined five new functions to the device driver interface :

- GreOpenScreenChangeArea
- GreGetScreenChangeArea
- GreCloseScreenChangeArea
- GreGetScreenBits
- GreSetScreenBits

The display driver must support all five functions. Currently, only the 32-bit VGA display driver can

do so. The 32-bit XGA display driver will provide support soon.

A screen change area describes where drawing has occurred on the screen. When a screen change area is opened by an application, any subsequent drawing order to the



Refreshing pixels with their current values still causes the bounding rectangle to be accumulated in the SCA.

display driver that results in at least one pixel being written on the screen will cause the bounding rectangle for that drawing order to be accumulated in the screen change area. Because this rectangle is accu-

Do these quotes sound familiar?

"It doesn't crash in the debugger!"

"Exactly, what did you do?"

"I can't reproduce it!"

"Where should I put WinGetLastError?"

"Why does WinDefWindowProc generate an error?"

"It must be a configuration problem!"

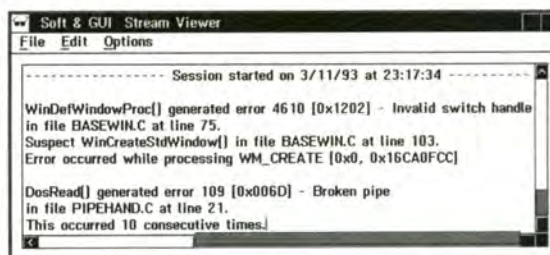
Introducing the API Debugger which gives Answers, not Guesses

Error Manager

32-bit
Version 2.0
for OS/2 2.X

Include 1 header file.
Link 1 DLL.
Set 1 environment variable.

- Finds intermittent errors without the Debug Kernel.
- Logs messages to a file, pipe or our PM viewer.
- Works with optimized code in realtime situations.
- Notifies your window procedure or callback function
- Supports both PM and Fullscreen programs
- Enables event-driven error handling
- Invaluable for Debug, QA Test and Production cycles



Includes a PM Viewer, Pointer Validity API and a Free copy of Command Line

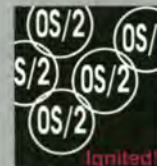
\$225



Soft & GUI

2224 East 21st Street
Brooklyn, New York 11229

(718) 769-8017





Processing a WM_PAINT message for a PM window

In this example a bitmap is the source of the data that is being displayed in the PM window's client area.

```
case WM_PAINT:

    /*****\
    * Obtain a cached PS and the rectangle to be repainted *
    \*****/
    hps = WinBeginPaint( hwndClient, NULL, &rcl );

    /*****\
    * Copy rectangle into a point array as used by GpiBitBlt *
    \*****/
    aptl[0].x = rcl.xLeft;
    aptl[0].y = rcl.yBottom;
    aptl[1].x = rcl.xRight;
    aptl[1].y = rcl.yTop;

    /*****\
    * Adjust source for scrolling *
    \*****/
    aptl[2].x = rcl.xLeft + ptlOrigin.x;
    aptl[2].y = rcl.yBottom + ptlOrigin.y;

    /*****\
    * Repaint the required client area *
    \*****/
    GpiBitBlt( hps, hpsMem, 3L, aptl, ROP_SRCCOPY, 0L );

    /*****\
    * Return the PS to the cache *
    \*****/
    WinEndPaint( hps );

    break;
```

Figure 2. Processing a WM_PAINT message for regular PM applications, minimizing traffic.

mulated after all clipping has been performed, it could be considerably smaller than any rectangle collected using traditional GPI bounds accumulation. Further, since clipping operations could render a single order drawn at the GPI level to several clipped orders, the bounding

rectangles of each of the clipped orders would then be accumulated in the screen change area. Even refreshing pixels with their current values still causes the bounding rectangle to be accumulated in the screen change area.

An application can query its

screen change area at any time, which causes the screen change area to be reset to null and ready to record the location of all subsequent drawing to the screen. The screen change area is then combined with a graphics region provided by the application. When an application no longer requires a screen change area, it must close the area to free unwanted resources. Several screen change areas can be open simultaneously.

Once an application has defined the drawing region via screen coordinates, it can obtain the pixel data defined within that region. Again, OS/2 2.0 helps with two new functions, GreGetScreenBits and GreSetScreenBits. Using GreGetScreenBits, the application provides a region handle, a pointer to available memory, and the amount of memory available. The display driver obtains and compresses the pixel data, storing it beginning at the address provided by the application and ensuring that it does not exceed the space allotted. The region is then edited to define the area that was not stored so the application can try again with another buffer if it chooses. DCAF uses this to ensure it can cover the entire screen area that has been drawn to.

The buffer is sent to the monitoring workstation, which uses GreSetScreenBits to unpack the buffer into a bitmap. From there, it can display the bitmap in a window and facilitate other operations such as clipping and scrolling.

HOOKS ASSISTANCE

DCAF also intercepts functions to the display driver with the new engine and display hooks (provided by the 32-bit graphics engine shipped with OS/2 version 2). DCAF's 32-bit hooks dynamic link library is called during boot time with a pointer to the display driver

Announcing TLIB 5.0!TM Version Control for DOS & OS/2

- The experts loved TLIB 4:

"...amazingly fast... TLIB is a great system." **PC Tech Journal**
 "TLIB has features and power to spare... TLIB is easy to use and the fastest of the reviewed packages." **Computer Language**
 "I will not program without it." **Uptime Magazine**

- Now TLIB 5.0 adds:

Automatic branching. Automatic version labeling across branches. Language-independent preprocessor, for "conditional compilation" in languages which do not support it (like dBase). N-way-tree version numbers. Branch and whole-library locking. OS/2 support.

- Plus the features they loved in TLIB 4:

Check-in/out locking. Branching, for parallel development. Keywords. Full binary file support (does not depend upon CRs in the file like other products). Wildcard and list-of-file support; can create lists by scanning source code for includes. Can merge (reconcile) multiple simultaneous changes and undo intermediate revisions. Network and WORM optical disk support. Mainframe-compatible delta generator for Pansophic, ADR, IBM, Sperry formats. Includes integrated PD MAKE by L. Dyer; also integrates with OpusTM MAKE, SlickTM MAKE, others.

MS-DOS \$139, OS/2 (with MS-DOS) \$195 + shipping.
 5 station network: MS-DOS \$419, OS/2 \$595. Call for other sizes.



Burton Systems Software

PO Box 4156, Cary, NC 27519 (919) 233-8128

Magus View – The PostScript Breakthrough!

Magus ViewTM – The revolutionary new PostScript-language rendering library from Magus. PostScript is the universal language for the device-independent representation of graphics and text; virtually all business software can generate PostScript output. Now *you* can create programs which display or print documents from any source, without need for the originating application, proprietary file formats, or a PostScript printer.

- Create front ends for document imaging systems – display PostScript files, or use PostScript as the image definition language
- Enhance collaborative applications such as electronic mail or other "groupware" – support documents with complex graphics and fonts
- Create host-based PostScript drivers for non-PostScript printers
- Bring a new level of fidelity to print-previewing in your applications

Magus View is available in DLL form for OS/2 2.0 and Microsoft Windows 3.1. Programming interfaces are provided for C, Smalltalk, and Digitalk's PARTS Workbench. Prices start at \$495 for a single Magus View Developer's Kit.

MAGUS

PO Box 390965 • Mountain View CA 94039-0965 • USA
 (800)848-8037 • (415)940-1109 • sales@magus.com

YOU WOULDN'T THINK ABOUT REINVENTING THE WHEEL... WHY REINVENT REASONING?



Take a quantum leap forward with The Integrated Reasoning Shell (TIRS). TIRS offers:



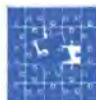
Speed

Fast performance with 32-bit processing



Productivity

Reduce development and maintenance time



Embeddability

Fully integrates with your application



Flexibility

Do it your way- Use your favorite DBMS

TIRS features include multiple platforms (OS/2, AIX/6000, VM, and MVS), frame-based reasoning with inheritance, a code generator (C and PL/I), and a GUI development environment.

For more information about TIRS and our two month product trial period, call:

1-800-IBM-SOLV

Outside the US: (408) 463-4381

IBM
 © IBM Corp. 1993. ©

IBM, TIRS, OS/2, and AIX/6000 are trademarks of the International Business Machines Corporation.


```

case WM_PAINT:
{
    HPS          hps;          /* cached micro ps          */
    USHORT       attr;         /* saved AVIO cursor attributes */
    VIOCURSORINFO vioci;       /* VIO cursor info          */
    RECTL        rcl;          /* Repaint rectangle        */
    SHORT        sHeight;      /* AVIO PS's character cell size*/
    SHORT        sWidth;       /* AVIO PS's character cell size*/
    SWP           swp;          /* Window position, dimensions */
    POINTL        ptlOrigin;    /* Repaint rectangle origin    */
    SHORT         sRowOrigin;   /* AVIO PS's origin in cell C.S.*/
    SHORT         sColOrigin;   /* AVIO PS's origin in cell C.S.*/
    USHORT        usOffset;     /* offset of first cell to show */
    SHORT         cxBorder;     /* width of window borders    */

    /*****\
    * Hide the cursor *
    \*****/
    attr = vioci.attr;
    vioci.attr = 0xFFFF;
    VioSetCurType( (PVIOCURSORINFO)&vioci, hvpsClient );
    vioci.attr = attr;

    /*****\
    * Obtain a cached PS and the rectangle to be repainted *
    \*****/
    hps = WinBeginPaint( hwndClient, NULL, &rcl );

    /*****\
    * Determine origin (inclusive) of repaint rectangle in pixels. *
    * Note: The AVIO co-ordinate system places origin at top left. *
    \*****/
    WinQueryWindowPos( hwndClient, &swp );
    ptlOrigin.y = swp.cy - rcl.yTop;
    ptlOrigin.x = rcl.xLeft;

    /*****\
    * Convert origin from units of pixels to character cells.      *
    * Note: top and left edge of window is character cell aligned. *
    \*****/
    VioGetDeviceCellSize( &sHeight, &sWidth, hvpsClient );
    ptlOrigin.y = ptlOrigin.y / sHeight;
    ptlOrigin.x = ptlOrigin.x / sWidth;

    /*****\
    * Add in the AVIO PS's own character based origin *
    \*****/
    VioGetOrg( &sRowOrigin, &sColOrigin, hvpsClient );
    ptlOrigin.x += sColOrigin;
    ptlOrigin.y += sRowOrigin;

    /*****\
    * Calculate the offset of the top left cell to be redrawn. Use
    * width specified when AVIO PS was created. Note: offset is in character cells.
    \*****/
    usOffset = (USHORT)(ptlOrigin.y * sVPSWidth + ptlOrigin.x);

```

Figure 3. Processing a WM_PAINT message for an AVIO window (continued on page 45)



```

/*****\
* Get the dimensions of the rectangle in cells. Note: peculiar effect of window size to be accounted for.
\*****/
cxBorder = (SHORT) WinQuerySysValue( HWND_DESKTOP
                                   , SV_CXSIZEBORDER
                                   );
sHeight = (SHORT)(rc1.yTop - rc1.yBottom + sHeight - 1
                  + sHeight - cxBorder)
          / sHeight;
sWidth = (SHORT)(rc1.xRight - rc1.xLeft + sWidth - 1
                 + sWidth - cxBorder)
         / sWidth;

/*****\
* Repaint the required client area *
\*****/
VioShowPS( sHeight, sWidth, usOffset, hvpsClient );

/*****\
* Return the PS to the cache *
\*****/
WinEndPaint(hps);
}
break;

```

Figure 3. Processing a WM_PAINT message for an AVIO window (continued from page 44)



Let Gpfs write the GUI you design

Using the powerful point and click visual programming environment of Gpfs*, you can prototype, test and generate a complete OS/2 PM GUI in a few hours or days rather than the weeks or months required to hand code the same design. Even a relatively simple GUI can require writing thousands of lines of code, but with Gpfs you simply draw your user interface on the screen. The integrated dialogue editor of Gpfs permits actions and context sensitive help to be linked to controls as you create them. Gpfs then generates error free ANSI C complete with embedded SQL statements.

Gpfs is optimized to take full advantage of OS/2 PM, the most powerful and robust GUI system available. Since Gpfs code directly accesses the PM API, there is no run time module to distribute with your application and no added overhead or royalties.

Gpfs keeps the entire design definition in one file. This permits easy re-entry for updates as well as regeneration for different targets. 32 bit OS/2, 16 bit OS/2 and MS Windows code generation.

Gpfs supports:

- Simple and direct linkage of the interface to program logic, built in or user defined functions.
- Direct association of help screens with controls, and complete integration into the PM Help Presentation Facility.
- Flexible use of Presentation objects (fonts, colors, etc.) with controls and windows (client area and frame).
- Simple inclusion of bitmaps for use on About screens, user-defined buttons, and menu or pulldown entries.
- Automatic embedded SQL statements to read OS/2 DataBase Manager tables, directly into combo or list boxes.
- Multi-thread programming.
- Multiple source file generation.
- Automatic creation of controls that scale with window size.
- Inclusion of user defined controls

Try us out

Order Gpfs today for just \$995.⁹⁹

Call Gpfs Systems Inc. at:

(203) 873-3300 or (800) 831-0017 - fax (203) 873-3302. Free demo software available
30 Falls Rd., P.O. Box 414, Moodus, CT 06469

* GUI Programming Facility

Gpfs 2.0

Available Now
32 Bit Code Generation
CUA '91 Controls



dispatch table. The table contains the addresses of functions that implement drawing orders (among other things). At this point, DCAF can overwrite the dispatch table entries with its own after saving the original addresses, intercepting any drawing orders in the dispatch table and noting the parameters so they can be replayed on the monitoring workstation. (Every hook must call the original function so as not to affect the operation of the workstation being monitored.)

Users may test competitive applications to see which perform better when accessed remotely.

OPTIMIZING FOR REMOTE PERFORMANCE

The user can do several things to minimize traffic. For example, if a help desk is contacted to help with a particular application, it makes sense to minimize or close all unrelated windows drawing to the screen. The subject application can also help by considering its drawing operations during remote viewing.

To increase application performance during remote access, the application must cause less traffic to be sent across the network. There are three basic guidelines for developers regarding this traffic:

1. Do not use more screen real estate than is required. If you are only going to display two lines of text, why put them in a window that takes up half the screen?

2. When using color, avoid using shades that are very similar. On slower networks, the remoting software may reduce a 256-color palette to 16 colors, causing similar colors to appear as the same color when viewed remotely.

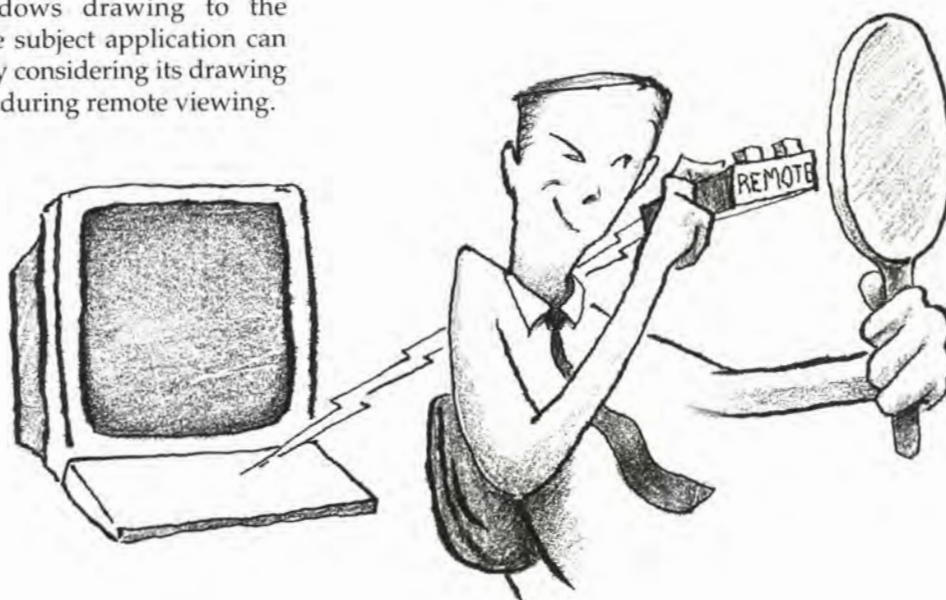
3. Do not draw unnecessarily. When your window procedure receives a `WM_PAINT` message, repaint only the area defined in the rectangle provided. When invalidating part of a window to cause a `WM_PAINT` message, invalidate only the minimum area necessary. If several areas need to be invalidated at once, consider whether it is better to invalidate each individually—allowing the repaint to occur for each individual area—rather than invalidat-

ing the bounding rectangle for the whole area. (It is unnecessary to worry about which parts of the window are visible for DCAF; all areas are accumulated in the screen change area after clipping.)

Figures 2 and 3 are examples of how to repaint the minimum area required, with Figure 2 showing the procedure for full PM windows (the most common case) and Figure 3 for AVIO windows (a more difficult situation).

In Figure 3, an AVIO presentation space is the source of the data displayed in the PM window's client area. Note that the variable declarations shown are for local variables only; this is not intended to be a complete list of the variables required.

Greg Loten, 200 W. 60th St., New York, N.Y. 10023. Loten joined IBM as a PC programmer in 1985, working on graphics software including PM and its applications. Recently, he worked on adding PM support to IBM's Distributed Console Access Facility. He now works as a freelance programmer. Loten received a B.Sc. in computer science from Imperial College, University of London.



And
the
winner
is...

AM

BEST
OS/2 EXPLOITATION

BEST
PRODUCTIVITY GAINS

BEST
DATABASE INTEGRATION

BEST
CUSTOMER SATISFACTION

BEST
ADVANCED CLIENT/SERVER TOOL



Intelligent Environments
USA 508•640•1080 ♦ EUROPE +44•932•772266

•BOSTON•CHICAGO•HOUSTON•LOS ANGELES•LONDON•FRANKFURT•STOCKHOLM•

INTELLIGENT ENVIRONMENTS INC., 2 HIGHWOOD DRIVE, TUESBURY, MA. ALL PRODUCTS AND PRODUCT NAMES MENTIONED ARE REGISTERED TRADEMARKS OF THEIR RESPECTIVE COMPANIES.





GUI

The container control provides application developers with many functions. During its development and beta test cycle, many customers requested information on how to perform certain tasks using the container. This article covers some commonly asked questions about areas of the container. **BY PETER HAGGAR and PETER BRIGHTBILL**

Programming the OS/2 Container Control: By Example

THE OS/2 CONTAINER CONTROL WAS shipped with OS/2 2.0 as a 32-bit control and with the CUA Controls Library (CCL/2) as a 16-bit control; the two are functionally equivalent within their respective products, and the information in this article can be used by developers working with either product. This article assumes the reader has some knowledge of the container control and how to program to it. For more information on the container

control, see the References section at the end of this article.

MOVING THE SPLITBAR WITH THE KEYBOARD

While the container control allows users to move the splitbar (the vertical bar that separates two windows in the details view, shown in Figures 5 and 6) using a pointing device such as a mouse, many developers have asked how to do so using the



Peter Haggard



Peter Brightbill

```
/* This function will track the splitbar in details view using the
 * keyboard. This function could be called as a result of a menu
 * item selected by the user. */
VOID TrackSplitbar (HWND hwndCnr)
{
    TRACKINFO ti;
    RECTL rclDV, rclCnr, rclCnrTitle;
    CNRINFO CnrInfo;

    /* Query the CnrInfo structure from the container. From it we will
     * get the current position of the splitbar. */
    WinSendMsg (hwndCnr, CM_QUERYCNRINFO,
        MPFROMP(&CnrInfo), MPFROMSHORT(sizeof(CNRINFO)));

    /* Get the window rectangles for the container title window, the left
     * details view window, and the container window itself. */
    WinQueryWindowRect (WinWindowFromID (hwndCnr, CID_CNRTITLEWND), &rclCnrTitle);
    WinQueryWindowRect (WinWindowFromID (hwndCnr, CID_LEFTDVWND), &rclDV);
    WinQueryWindowRect (hwndCnr, &rclCnr);
}
```

Figure 1. Adding keyboard manipulation of the splitbar (continued on page 49)



```
/* Initialize the TrackInfo structure. The splitbar has the width of
 * a SizeBorder and the height of the container window minus the
 * container title window. */
ti.cxBorder = (SHORT)WinQuerySysValue (HWND_DESKTOP, SV_CXSIZEBORDER);
ti.cyBorder = (SHORT)(rc1Cnr.yTop - rc1CnrTitle.yTop);
ti.cxGrid   = (SHORT)rc1Cnr.xRight;
ti.cyGrid   = (SHORT)rc1Cnr.yTop;

/* Set the x Keyboard increment to be the number of pels you want the
 * splitbar to move when the user presses the right and left arrow
 * keys. Set the y Keyboard increment to 0. */
ti.cxKeyboard = 10;
ti.cyKeyboard = 0;

/* Set up the rectangle to be tracked, the rectangle of the splitbar.*/
ti.rc1Track.xLeft  = CnrInfo.xVertSplitbar;
ti.rc1Track.xRight = ti.rc1Track.xLeft + ti.cxBorder;
ti.rc1Track.yBottom = rc1Cnr.yBottom;
ti.rc1Track.yTop   = ti.cyBorder;

/* Set up the absolute boundary rectangle, the rectangle specifying
 * the boundary the tracking rectangle cannot exceed. Set it to be the
 * left, right, and bottom of the container and the top of the splitbar. */
ti.rc1Boundary.xLeft  = rc1Cnr.xLeft;
ti.rc1Boundary.xRight = rc1Cnr.xRight;
ti.rc1Boundary.yBottom = rc1Cnr.yBottom;
ti.rc1Boundary.yTop   = ti.cyBorder;
ti.pt1MinTrackSize.x  = ti.cxBorder;
ti.pt1MinTrackSize.y  = ti.cyBorder;
ti.pt1MaxTrackSize.x  = ti.cxBorder;
ti.pt1MaxTrackSize.y  = ti.cyBorder;

/* Set the track flags. TF_SETPOINTERPOS will put the mouse pointer
 * in the middle of the tracking rectangle (splitbar). */
ti.fs = TF_MOVE | TF_ALLINBOUNDARY | TF_SETPOINTERPOS;

/* If tracking is successful, inform the container of the new
 * splitbar position. */
if (WinTrackRect (hwndCnr, NULL, &ti))
{
    /* Inform the container of the new splitbar position. */
    CnrInfo.xVertSplitbar = ti.rc1Track.xLeft;
    WinSendMsg (hwndCnr, CM_SETCNRINFO,
                &CnrInfo, MPFROMLONG(CMA_XVERTSPLITBAR));
}
return;
}
```

Figure 1. Adding keyboard manipulation of the splitbar (continued from page 48)



```

/* This function is used to bring a particular record in the container
 * to the viewport. It attempts to place the record as close to the
 * top left corner as possible. */
VOID ScrollToRecord (HWND hwndCnr, PRECORDCORE pRecord)
{
    RECTL          rclViewport, rclItem;
    QUERYRECORDRECT QueryRecordRect;
    LONG           lMargin = 4L;
    CNRINFO        CnrInfo;

    /* Query the container for the current view. If a text view is the
     * current view, then CMA_ICON is invalid for QueryRecordRect.fsExtent */
    WinSendMsg (hwndCnr, CM_QUERYCNRINFO,
                MPFROMP(&CnrInfo), MPFROMSHORT(sizeof(CNRINFO)));

    /* Query the container for the position of the record relative
     * to the viewport. We want the rectangle containing both the icon
     * and the text of the record. */
    QueryRecordRect.cb      = sizeof(QUERYRECORDRECT);
    QueryRecordRect.pRecord = pRecord;
    QueryRecordRect.fsExtent = CMA_ICON | CMA_TEXT;
    QueryRecordRect.fRightSplitWindow = FALSE;
    if (CnrInfo.flWindowAttr & CV_TEXT)
        QueryRecordRect.fsExtent = CMA_TEXT;
    else
        QueryRecordRect.fsExtent = CMA_ICON | CMA_TEXT;

    WinSendMsg (hwndCnr, CM_QUERYRECORDRECT,
                MPFROMP(&rclItem), MPFROMP(&QueryRecordRect));

    /* Query the container for the size of the current viewport.
     * This is necessary because the position of the record
     * is relative to the bottom left corner of the viewport. */
    WinSendMsg (hwndCnr, CM_QUERYVIEWPORTRECT,
                MPFROMP(&rclViewport), MPFROM2SHORT(CMA_WINDOW, FALSE));

    /* Scroll the container vertically to bring the record to a n
     * position just below the top of the viewport. */
    WinSendMsg (hwndCnr, CM_SCROLLWINDOW, MPFROMSHORT(CMA_VERTICAL),
                MPFROMLONG(rclViewport.yTop - rclItem.yTop - lMargin));

    /* Scroll the container horizontally to bring the record to a
     * position just to the right of the left edge of the viewport. */
    WinSendMsg (hwndCnr, CM_SCROLLWINDOW, MPFROMSHORT(CMA_HORIZONTAL),
                MPFROMLONG(rclItem.xLeft - lMargin));

    return;
}

```

Figure 2. Scrolling a record into the viewport

keyboard. Currently, the container only provides mouse manipulation of the splitbar. Figure 1 shows how to add this feature to the container. The CUA 1991 recommendation for initiating this function is to provide a **Split** entry in the system menu.

SCROLLING TO A PARTICULAR RECORD

With some applications, users may want to scroll a particular record into view. For example, a user might want to scroll the censored item to the top-left corner of the current viewport. The application can do this with the container control's messages. Important considerations include the position of the record relative to the current viewport and the size of that viewport. The container control

provides a number of useful query messages. Figure 2 shows a function that uses some of these query functions to make this scrolling feature possible. While this function will bring a record into view, it is not always possible to bring it to the top-left corner.

FILTERING RECORDS

The filter feature lets users see a subset of the objects in a container. The code in Figure 3 deals with a container of vehicles; the vehicles can be either gas powered (Mazda Miata) or manually powered (Schwinn Bicycle). The objects are displayed in details view and information about them is listed in the **Year** through **Mileage** fields in the **VEHICLERECORD** structure. Because not all columns apply to all types of vehicles, some fields are empty.

The code example allows a user

to filter the container for either gas-powered or manual vehicles. To accomplish this, it is necessary to create a filter function (**pfnFilter**, shown in Figure 4) that is called by the container once filtering is initiated with the **CM_FILTER** message. The **bGasPowered** field of each **VEHICLERECORD** has been set to indicate the type of each vehicle. The container control allows the application to pass a pointer as **mp2** of the **CM_FILTER** message, which is then passed to the application's filter function as the second parameter. In Figure 4, the **FILTERSTRUCT** structure is passed for this purpose. This structure is set up differently depending on whether the user is filtering for gas-powered or manual vehicles. Once the **CM_FILTER** message is sent to the container, the **pfnFilter** function is called once for each item in the container. The **pfnFilter**

Expressly for the OS/2 Desktop and Workgroup

Relish exploits OS/2 to give you unmatched flexibility in coordinating and managing commitments. Enhancements for the Workplace Shell foster intuitive drag-and-drop manipulation of new and existing calendar entries.

Convenient to use. Print the schedule for any day just by dragging the date to your Workplace Shell printer... Reschedule a meeting by dragging it to a new date or time... It's really that easy.

Relish 32-Bit for personal calendaring brings the intuitive edge to desktop time management. Includes integrated To Do list and Phone Book. \$189.

Relish Net 32-Bit for LAN-based groups blends group and resource scheduling with all the personal features of Relish. Each 5 user increment \$685.

Coming soon... the Relish API. Allows information exchange between Relish and corporate applications. Perfect for scheduling follow-ups. Call for details.

Relish®

32-Bit Time Management



Flexible. Drag-and-drop scheduling and categorizing.
Reliable. Dynamic saving and refreshing across views.
Versatile. Automatic reminders and LAN reconciliation.
Cutting-Edge. Multi-threaded, client-server design.

Ready to spread some Relish? Call for a free taste test.
 32-bit for OS/2 2.0. 16-bit versions also available.

Sundial Systems Corporation (310) 596 • 5121
 909 Electric Avenue, Suite 204, Seal Beach, CA 90740 USA



```

/* User defined record structure.  Contains RECORDCORE plus application
 * specific fields. */
typedef struct _VEHICLERECORD {
    RECORDCORE    RecordCore;
    PSZ            Year;          /* Year of manufacture          */
    PSZ            Vehicle;       /* Make and Model              */
    ULONG          Price;         /* Current Value in U.S. Dollars */
    PSZ            Color;        /* Dominant Color               */
    ULONG          MPG;          /* Miles Per Gallon            */
    ULONG          CalPerHr;      /* Calories Per Hour           */
    ULONG          Mileage;       /* Current Mileage              */
    BOOL           bGasPowered;   /* TRUE->Gas Powered, FALSE->Manual*/
} VEHICLERECORD, FAR *PVEHICLERECORD;

/* Structure definition to set up return values for filter function */
typedef struct _FILTERSTRUCT {
    BOOL          RetVal1;
    BOOL          RetVal2;
} FILTERSTRUCT, FAR *PFILTERSTRUCT;

/* The User wants a subset of the container items(Process Filtering).
 * This code is called when a user selects menu options. */
case CNR_FILTER_GASPOWERED:
case CNR_FILTER_MANUAL:
{
    PFIELDINFO    pFieldInfo;
    FILTERSTRUCT  FilterStruct;

    /* Setup Return values.  TRUE->Visible, FALSE->NOT Visible
     * This filter function knows whether to filter out Gas Powered
     * objects or Manual Powered objects. Thus, we don't need two
     * separate filter functions. */
    if (SHORT1FROMMP(mp1) == CNR_FILTER_GASPOWERED)
    {
        FilterStruct.RetVal1 = TRUE;
        FilterStruct.RetVal2 = FALSE;
    }
    else
    {
        FilterStruct.RetVal1 = FALSE;
        FilterStruct.RetVal2 = TRUE;
    }

    /* Tell the container to start the Filtering Process. */
    WinSendMsg (hwndCnr, CM_FILTER,
                MPFROMP(pfnFilter), MPFROMP(&FilterStruct));

    /* Obtain pointer to the first Details View Column */
    pFieldInfo = (PFIELDINFO)WinSendMsg (hwndCnr, CM_QUERYDETAILFIELDINFO,
                                         NULL, MPFROMSHORT(CMA_FIRST));
}

```

Figure 3. Using the filter feature (continued on page 53)



```

/* Iterate through all columns and make visible only those columns
 * that apply to the desired subset. The pUserData field of the
 * FIELDINFO structure is used. */
while (pFieldInfo)
{
    if (SHORT1FROMMP(mp1) == CNR_FILTER_GASPOWERED)
    {
        /* If the desired subset is Gas Powered then make columns
         * labeled with a "2" invisible. "2" -> Manual Powered
         * columns only. */
        if (pFieldInfo->pUserData == (PVOID)2)
            pFieldInfo->f1Data |= CFA_INVISIBLE;
        else
            pFieldInfo->f1Data &= ~CFA_INVISIBLE;
    }
    else
    {
        /* If the desired subset is Manual then make the columns
         * labeled with a "1" invisible. "1" -> Gas Powered
         * column only. */
        if (pFieldInfo->pUserData == (PVOID)1)
            pFieldInfo->f1Data |= CFA_INVISIBLE;
        else
            pFieldInfo->f1Data &= ~CFA_INVISIBLE;
    }
    pFieldInfo = (PFIELDINFO)WinSendMsg (hwndCnr,
                                          CM_QUERYDETAILFIELDINFO,
                                          MPFROMP (pFieldInfo),
                                          MPFROMSHORT(CMA_NEXT));
}

/* Refresh the Details View completely. */
WinSendMsg (hwndCnr, CM_INVALIDATEDDETAILFIELDINFO, NULL, NULL);
}
break;

```

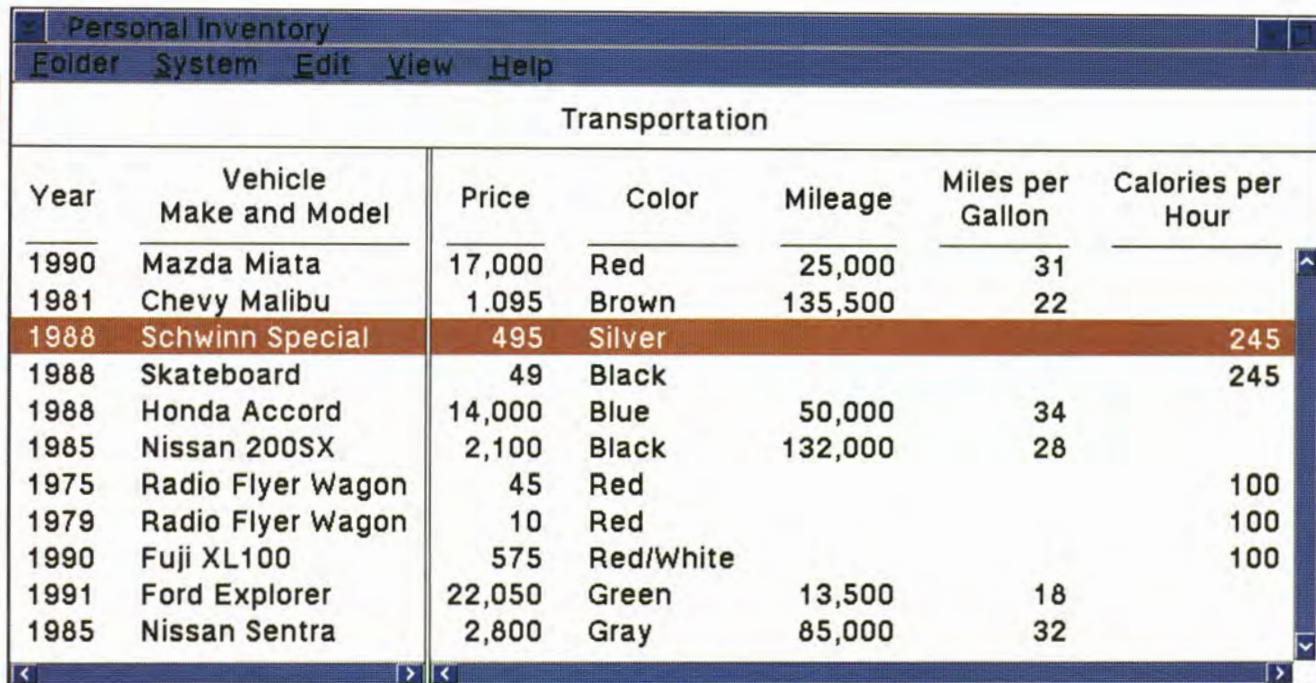
Figure 3. Using the filter feature (continued from page 52)

```

/* This function filters out either Gas-Powered or Manual vehicles
 * depending on the given return values in FILTERSTRUCT.
 * Using the bGasPowered field of VEHICLERECORD to distinguish. */
BOOL EXPENTRY pfnFilter(PRECORDCORE preccObj, PVOID pStorage)
{
    if (((PVEHICLERECORD)preccObj)->bGasPowered)
        return ((PFILTERSTRUCT)pStorage)->RetVal1;
    else
        return ((PFILTERSTRUCT)pStorage)->RetVal2;
}

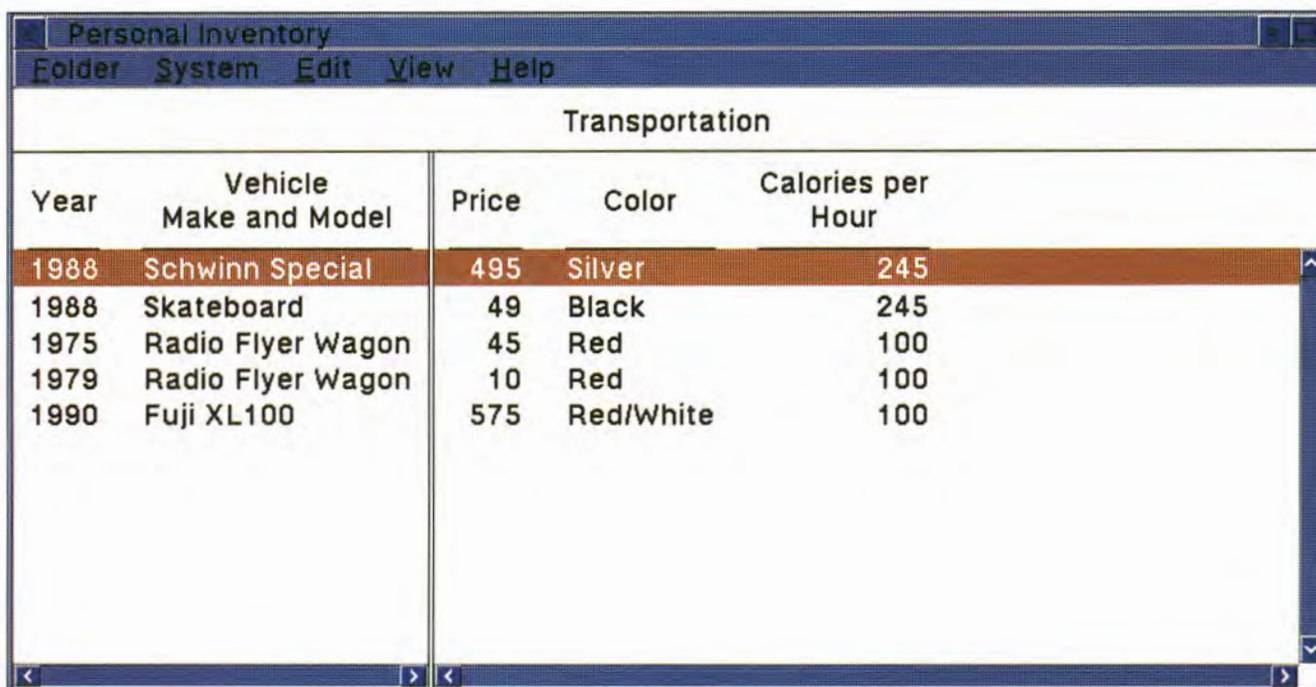
```

Figure 4. The filter function



Personal Inventory						
Folder System Edit View Help						
Transportation						
Year	Vehicle Make and Model	Price	Color	Mileage	Miles per Gallon	Calories per Hour
1990	Mazda Miata	17,000	Red	25,000	31	
1981	Chevy Malibu	1,095	Brown	135,500	22	
1988	Schwinn Special	495	Silver			245
1988	Skateboard	49	Black			245
1988	Honda Accord	14,000	Blue	50,000	34	
1985	Nissan 200SX	2,100	Black	132,000	28	
1975	Radio Flyer Wagon	45	Red			100
1979	Radio Flyer Wagon	10	Red			100
1990	Fuji XL100	575	Red/White			100
1991	Ford Explorer	22,050	Green	13,500	18	
1985	Nissan Sentra	2,800	Gray	85,000	32	

Figure 5. Container before filtering



Personal Inventory				
Folder System Edit View Help				
Transportation				
Year	Vehicle Make and Model	Price	Color	Calories per Hour
1988	Schwinn Special	495	Silver	245
1988	Skateboard	49	Black	245
1975	Radio Flyer Wagon	45	Red	100
1979	Radio Flyer Wagon	10	Red	100
1990	Fuji XL100	575	Red/White	100

Figure 6. Container after filtering for manually powered vehicles

function returns either TRUE (to make the item visible) or FALSE (to remove the item from the viewable subset).

Once the records are filtered, it is desirable to show only those columns that apply to the given subset. The MPG (miles per gallon) column, for example, is not

applicable to manually powered vehicles. The container control provides the CFA_INVISIBLE column attribute for this purpose. The pUserData field of the FIELDINFO data structure has been set to indicate the type of column; 0 is for all vehicles, 1 for gas powered only, 2 for

manual powered only. Iterate through all columns, turning the CFA_INVISIBLE attribute on or off depending on the desired subset. Finally, update the changes by sending the CM_INVALIDATEDDETAIL FIELDINFO message to the container. Figure 5 shows the container before

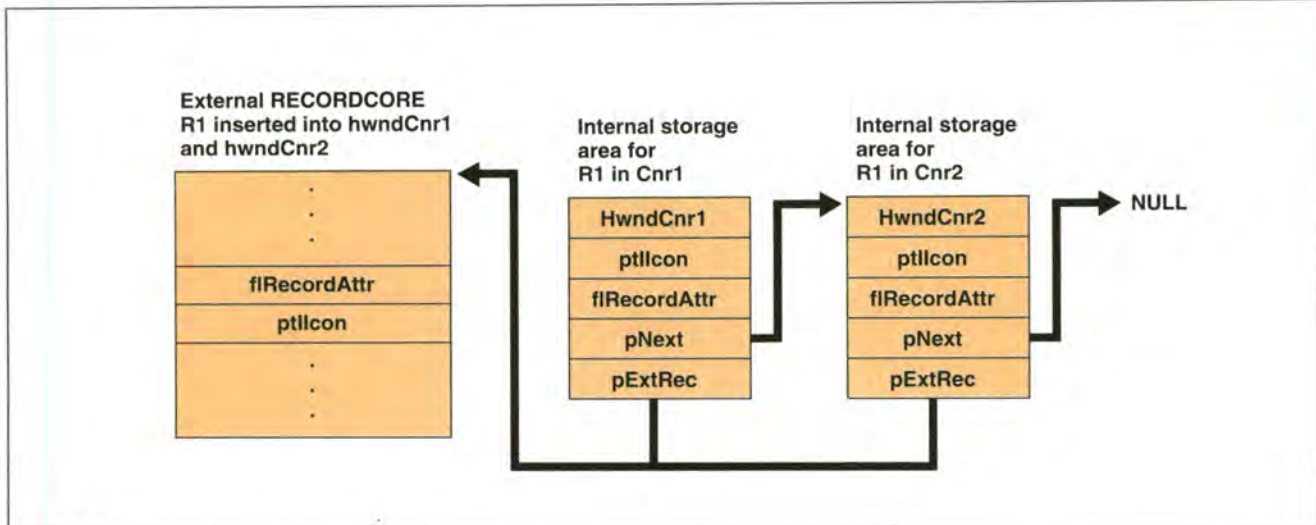


Figure 7. External and internal record memory layout

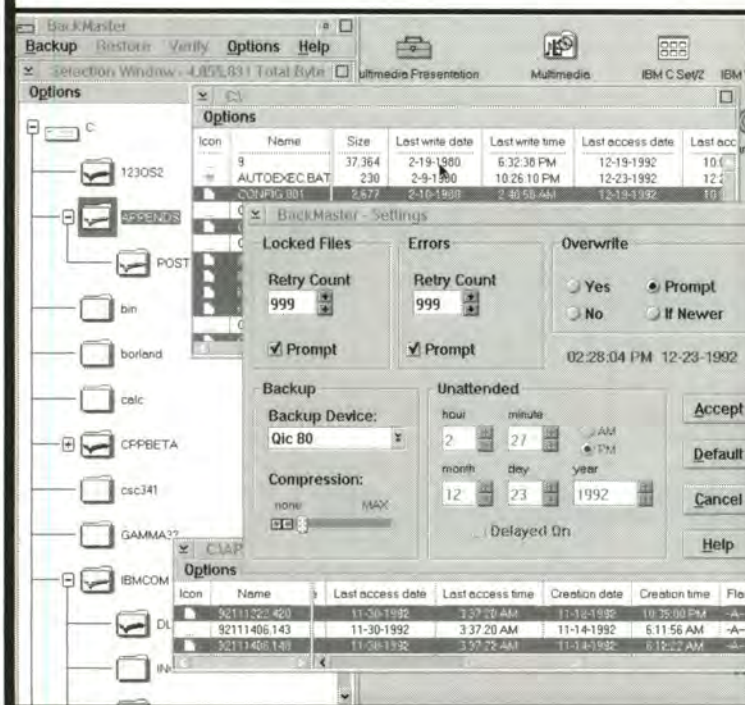
any filtering has been done. Figure 6 shows the container after it has been filtered for manually powered vehicles.

RECORD SHARING

The container's record sharing is probably one of its least understood features. Record sharing is the

ability of an application to allocate record memory once, then insert the same record into an unlimited number of container windows. The

BackMaster for OS/2 2.x



32-Bit Backup for OS/2

32-Bit Multi-Threaded PM Backup Utility.
Supports FAT and HPFS File Systems.
Handles Long Names, and Extended Attributes.
Backup WPS, Desktop, and System Files.
Total/Partial/Incremental/Unattended Backups.
Backup to Floppy Disk and QIC-40/80 Tape Drives.
Can Read QIC-40/80 DOS Tapes.
Easily Migrate your DOS files to OS/2 using Tape.
We Support CMS Jumbo & Generic QIC-40/80
Runs in the Background.
Uses Industry Standard STAC LZS Data Compression.

Only \$79.95

Texas Residents add 8 1/4% Sales Tax
VISA/MC/COD Welcome

Call our BBS for a Free Demo Version

MSR Development

Rt 7 #6409, Nacogdoches, TX 75961
Ph (409) 564-1862, BBS (409) 560-5970



```

/* This function will allocate a record and insert it into the 3
 * container windows passed in. After allocating the record, 3 unique (x,y)
 * positions, as well as 2 sets of unique attributes, will be assigned. */
BOOL AllocateAndInsertRecord(HWND hwndCnr1, HWND hwndCnr2,
                             HWND hwndCnr3, HPOINTER hIcon,
                             PSZ pszIconText, RECORDINSERT RecordInsert)
{
    PMINIRECORDCORE pRecord;
    LONG rc = 0;
    /* Allocate one MINIRECORDCORE structure from the first container.
     * Do not allocate any addition bytes of storage. */
    pRecord = (PMINIRECORDCORE)WinSendMessage (hwndCnr1, CM_ALLOCORECORD,
                                                MPFROMLONG(0), MPFROMSHORT(1));

    /* If we don't get a record, bail out now. */
    if (pRecord)
    {
        /* Fill in the record memory with the initial information. */
        pRecord->cb = sizeof(MINIRECORDCORE);
        pRecord->hptrIcon = hIcon;
        pRecord->ptlIcon.x = 100;
        pRecord->ptlIcon.y = 100;
        pRecord->flRecordAttr = CRA_SELECTED | CRA_RECORDREADONLY;
        pRecord->pszIcon = malloc(TEXT_SIZE);
        strcpy (pRecord->pszIcon, pszIconText);

        /* Insert the record into the first container. */
        rc = (LONG)WinSendMessage (hwndCnr1, CM_INSERTRECORD,
                                    MPFROMP(pRecord), MPFROMP(&RecordInsert));

        if (rc)
        {
            /* Change the (x,y) position of the record, and insert the same
             * record into the second container. */
            pRecord->ptlIcon.x += 50;
            pRecord->ptlIcon.y -= 30;

            rc = (LONG)WinSendMessage (hwndCnr2, CM_INSERTRECORD,
                                        MPFROMP(pRecord), MPFROMP(&RecordInsert));
        }

        if (rc)
        {
            /* Change the (x,y) position again, and turn the selection attribute
             * off of the same record and insert it into the third container.
             * If all is successful, return TRUE, otherwise return FALSE. */
            pRecord->ptlIcon.x = 275;
            pRecord->ptlIcon.y = pRecord->ptlIcon.x + 40;
            pRecord->flRecordAttr &= ~CRA_SELECTED;

            rc = (LONG)WinSendMessage (hwndCnr3, CM_INSERTRECORD,
                                        MPFROMP(pRecord), MPFROMP(&RecordInsert));
        }
    }

    /* Convert rc to BOOL and return. */
    return !!rc;
}

```

Figure 8. Allocating and inserting shared records

position and attributes of each record are kept separately on a per-container basis. For example, the same record can be inserted into multiple containers, with a different position and different attributes in each container. In this situation, the application would allocate memory for only one record while the container control manages data associated with that record across multiple containers. The records, although identical and occupying the same area of memory, can appear totally independent to the user. The record sharing feature, however, is available only to containers operating in the same process.

Record sharing is useful to an application that must display one record in multiple container windows at the same time. The OS/2 2.0 Workplace Shell, which uses the container control for all its folders, uses record sharing to implement CUA's Multiple Concurrent View concept. For example, a user can open a command-prompts folder simultaneously in the Icon and Details views. The objects in these folders, although appearing separately, are actually the same objects (i.e., the same container record inserted into multiple container windows). But because the positions and attributes of each object are unique to its container, they can appear to the user to be separate objects.

To take full advantage of record sharing in an application, the developer must code certain parts of the program carefully. Understanding how record sharing is implemented can help develop a well-behaved application.

As each record is inserted into a container using `CM_INSERTRECORD` or a currently inserted record is invalidated using `CM_INVALIDATERECORD`, the container copies the attributes of the `flRecordAttr` field and the (x,y) position specified in

the `ptIcon` field to an internal storage area for that record. As a record's attributes and position are changed, it is updated only in the internal storage area, not in the fields of the external record structure. Therefore, when the same record is inserted into multiple containers, its positions and attributes can vary, as they are stored internally for each record on a per-container basis. Figure 7 shows the memory layout resulting from the same record being inserted into two different container windows. The application has access only to the external record. The container has access to the internal storage area for each record, as well as to the external record.

When a record is returned to the application from the container, the `ptIcon` and `flRecordAttr` information is retrieved from the records' internal storage area and updated in the external record to reflect the state of the record in that particular container. For example, if the application uses the `CM_QUERY_RECORD` message, the `ptIcon` and `flRecordAttr` fields of the returned record will be updated to reflect the record's current state in the container from which it has been queried. If the application wishes to query information for a record, the `CM_QUERYRECORDINFO` message is sent to the appropriate container.

When writing an application, the most important aspect of record sharing is that `CM_INVALIDATERECORD` should almost always be preceded by `CM_QUERYRECORDINFO` for the records being invalidated. Because `CM_INVALIDATERECORD` will copy the external record's current `ptIcon` and `flRecordAttr` fields to the record's internal storage area, this information must be accurate. It is not necessary to send the `CM_QUERYRECORDINFO` message if you are explicitly setting the `ptIcon` position and `flRecordAttr` fields.

Figures 8 and 9 give code samples of record sharing.

When using record sharing, it is important to note that when an application attempts to free the memory of a record (`CM_FREERECORD` or `CM_REMOVERECORD` with the `CMA_FREE` flag), it will be freed only if the record is not inserted into any other container. For each version of the container with which record sharing is used, there are differences in behavior that affect application design. One major difference between the 16- and 32-bit containers involves the memory area from which records are allocated.

Record sharing is useful to an application that must display one record in multiple container windows at the same time.

Record Sharing with the 16-Bit Container. With the 16-bit CCL/2 container, all records are allocated from a common storage space created when the container is created. When a container is destroyed, so is the storage space for that container. The destruction of the storage space and all the records within it occurs regardless of whether the records are inserted into another container. Avoid allocating records from one container (A), inserting the same records into another container, then destroying A before removing the records from the other container. In this case, the records allocated from A would be freed,





```

/* This function will change a specified record's position and/or
 * attributes for a particular container. */
BOOL ChangeRecordPosAndAttr(HWND hwndCnrChange, PMINIRECORDCORE pRecord,
                             LONG DeltaX, LONG DeltaY, LONG NewAttrs)
{
    USHORT usFlag = 0;

    /* Query the current information for the pRecord that is contained in
     * hwndCnrChange. */
    if (WinSendMsg (hwndCnrChange, CM_QUERYRECORDINFO,
                    MPFROMP(&pRecord), MPFROMSHORT(1)))
    {
        /* Check to see if the new attributes are indeed different than the
         * current record attributes. If they are different, set the
         * attributes and the minimum invalidate flag necessary. */
        if (pRecord->flRecordAttr != NewAttrs)
        {
            pRecord->flRecordAttr = NewAttrs;
            usFlag = CMA_NOREPOSITION;
        }

        /* Make sure at least one of the delta amounts is not 0. If so,
         * change the current (x,y) by the delta amount and reset the
         * invalidate flag such that the record will go to its new position. */
        if (DeltaX || DeltaY)
        {
            pRecord->ptIcon.x += DeltaX;
            pRecord->ptIcon.y += DeltaY;
            usFlag = CMA_REPOSITION;
        }

        /* If usFlag is set, a change has taken place with the record.
         * Erase the record and invalidate it at its new position. */
        if (usFlag)
        {
            WinSendMsg (hwndCnrChange, CM_ERASERECORD, MPFROMP(pRecord), NULL);
            return ((BOOL)WinSendMsg (hwndCnrChange, CM_INVALIDATERECORD,
                                     MPFROMP(&pRecord), MPFROM2SHORT(1, usFlag)));
        }
        else
        /* No error existed, but the record did not change either. */
        return (TRUE);
    }

    /* CM_QUERYRECORDINFO returned FALSE indicating that the given record
     * does not exist in the container specified by hwndCnrChange. */
    return (FALSE);
}

```

Figure 9. Updating shared records

with unpredictable results for other containers holding the same records.

It is important that records be freed only from the container from which they were allocated. Freeing a record from another container can lead to memory corruption. For example, a record is allocated from container A and then inserted into container B. B frees the record with `CM_FREERECORD`. This causes a memory error because the freed record was not part of B's storage space. The container checks only if a record is not inserted into any other container before it allows it to be freed.

These problems can be avoided by creating an invisible "source" container. Each record should be allocated from, and inserted into, the source container so no other container can free it. When the

application is terminated, destroy the invisible source container last. (When a container is destroyed, it will also free the memory of all `FIELDINFO` structures inserted into it.

Record Sharing with the 32-Bit Container. With the 32-bit OS/2 2.0 container, all records are allocated from the process address space, making them common across the process. When a container is destroyed, all records inserted into that container not inserted into any other container are automatically freed. Records can be allocated from one container and safely freed from another, since there is no concept of unique storage spaces for each individual container, as with the 16-bit container. Because of the way the 32-bit container manages memory, there is no need here for the concept of a source container.

SUMMARY

The container control provides developers with a variety of powerful functions. The examples presented here should make using the container control much easier.

The source code for this article, plus two sample container programs by the authors, can be found on the following electronic sources:

On CompuServe, the source code is in Library 13 of the OS2DF2 forum, under the name CNREXM. The sample programs are in Library 2 of the OS2DF1 forum, under the name CNRSAM.

In the U.S., call the IBM Technical Support Group PCC bulletin board service at (404) 835-6600. The source code and sample programs are in file area 11 (BYEXAMPL.ZIP and CNRSAMPL.ZIP, respectively).

ON IBM VM, issue the following commands:

```
REQUEST BYEXAMPL FROM HAGGAR AT
CARVM3
REQUEST CNRSAMP FROM HAGGAR AT
CARVM3
```

Peter Brightbill, IBM Programming Systems Laboratory, 11000 Regency Pkwy., Cary, N.C. 27512. Brightbill is a senior associate programmer in the IBM OS/2 PM Extensions department. He joined IBM in 1989 and has been working on OS/2 PM development since that time. He was a member of the container control development team and is currently working on C++ user interface class libraries. Brightbill received a B.S. in mathematics and computer science from Millersville University, Pa.

Peter Haggar, IBM Programming Systems Laboratory, 11000 Regency Pkwy., Cary, N.C. 27512. Haggar is a staff programmer in OS/2 PM extensions development. He joined IBM in 1987 and has been working in OS/2 PM development since 1989. Haggar was a member of the OS/2 container control development team and is currently working on C++ user interface class libraries. He received a B.S. in computer science from Clarkson University, N.Y.



REFERENCES

Haggar, Peter and Peter Brightbill. "Programming the OS/2 Container Control: The Basics," *IBM OS/2 Developer*. (Winter 1993): 96-101.

Haggar, Peter, Tai Woo Nam, and Ruth Anne Taylor. "Container Control: Implementing the Workplace Model," *IBM Personal Systems Developer*. (Winter 1992): 48-54.

OS/2 2.0 Programmers Guide Volume II (IBM Doc. S10G-6494)

OS/2 2.0 Presentation Manager Programming Reference Volume III (IBM Doc. S10G-6272)

SAA CUA Guide to User Interface Design (IBM Doc. SC34-4289)

SAA CUA Advanced Interface Design Reference (IBM Doc. SC34-4290)



Object-Oriented Programming

This article describes, through examples, how to create a program that can exist in and take advantage of the Workplace environment, yet avoid the problems posed by the single OS/2 process.

BY RICHARD REDPATH, JOE COULOMBE, and SUE HENSHAW

Workplace Shell Programming Using Multiple Processes



Richard Redpath



Joe Coulombe



Sue Henshaw

THE OS/2 2.0 WORKPLACE SHELL™ AND the objects it launches normally operate within a single OS/2 process. This can create problems related to reliability, security, database access, file access, and input or output redirection. This article describes, through examples, how to create a program that can exist in and take advantage of the Workplace environment, yet avoid the problems posed by the single OS/2 process.

The examples shown here demonstrate the steps required for developing a solution to the single process problem in the Workplace Shell. For the sake of brevity, these examples result in an application-oriented program rather than an object-oriented program that could take full advantage of the Workplace environment and CUA™ 91 user interface guidelines.

OS/2 2.0 WORKPLACE SHELL AND THE SINGLE-PROCESS PROBLEM

The Workplace Shell, the user interface to the OS/2 2.0 operating system, is designed to be an object-oriented user environment in which users can focus on work documents (called objects) rather than on application programs. The System Object Model (SOM) provides a class-oriented programming and run-time environment upon which the Workplace Shell's object-oriented user environment operates.

Because new classes are written as loadable dynamic link libraries, or DLLs, it can be useful to consider classes and DLLs to be the same. New classes are defined from parent classes, from which they may inherit various attributes. When a user requests or activates an instance of a class, the Workplace Shell loads the correct DLL and executes the methods it contains within the same single OS/2 process.

All these instances and classes operating within a single process can lead to several problems. Because all classes run in the same process, the crash of one can lead to the crash of all others currently running. In addition, the Workplace Shell's single address space can be easily violated across programs. Redirected input and output, directory settings, and other environmental settings can change between programs without warning, creating uncertainty for the programmer and the environment. Finally, the session limits on databases may reduce the number of instances that can access a given database.

In summary, using a single process for the logical operation of many objects poses several single-process problems. To get around these problems, you must create object classes in a way that minimizes exposure to problems and allows the removal of the self-imposed limitations when they are no longer needed. This article illustrates such a solution for the current Workplace environment.

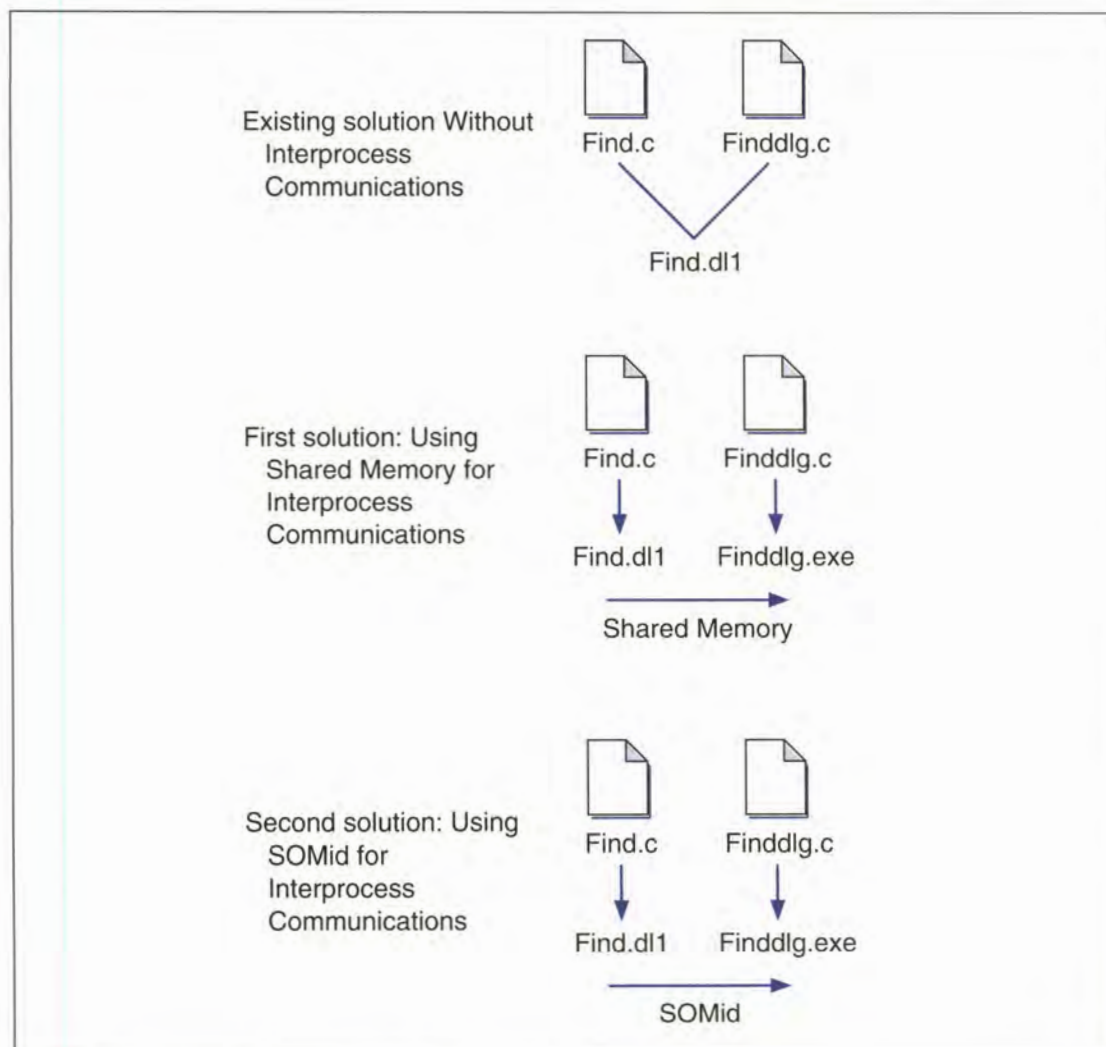


Figure 1. Comparison of three methods of programming

OVERVIEW OF THE SOLUTION

The best way to work around the shell single-process problem is to use interprocess communication between an object running inside the shell process and a process running completely outside it that supports the shell environment. This workaround consists of two parts.

Object Stub. The object stub is created for and executes entirely within the shell's single process and the SOM class hierarchy. It does little meaningful work, but rather spawns a

separate process in which the real work of the object can be done. Any messages received by the object stub are passed directly to the external process.

External process. The object stub launches an executable program, which is executed in a process completely separate from that of the Workplace Shell. Because the executable program performs all meaningful work for the object class, that work can be done apart from the influence of other objects and without endangering the Workplace Shell.



EXISTING PROGRAMS: OPERATING WITHIN A SINGLE PROCESS

This example describes the implementation of a standard program that operates entirely within the Workplace Shell's single process.

In this example, an object's standard pop-up menu is modified to add a Find string choice, which calls the `_wpMenuItemSelected()` method with the appropriate selection ID number. From this choice, an object window message is posted to display the dialogue contained in the `FINDDL.G.C` file along with a string search function as shown in Figure 2.

Define the Class. The `FIND.CSC` file defines the class object. The Find class is defined as a subclass of `WPAbstract`. An instance of this class can be created as a Workplace object. The methods overridden are `wpInitData`, `wpDelete`, `wpModifyPopupMenu`, and `wpNewItemSelected`.

Create the FIND.C Template Body. Also prepare a blank `.C` file into which the SOM compiler will place the

output of the compilation of the `.CSC` file. This includes, in addition to the standard include files, `find-def.h`, `find.h`, `find.ph`, and `finddat.h`. Because we are using SOM, we must also include the statement `VOID SOMLINK SOMInitModule(void)` and the

statement `FindNewClass(Find_MajorVersion, Find_MinorVersion)`.

Prepare a Make File. The standard make file for building the Find class object is used. This file invokes the SOM compiler, which inserts code

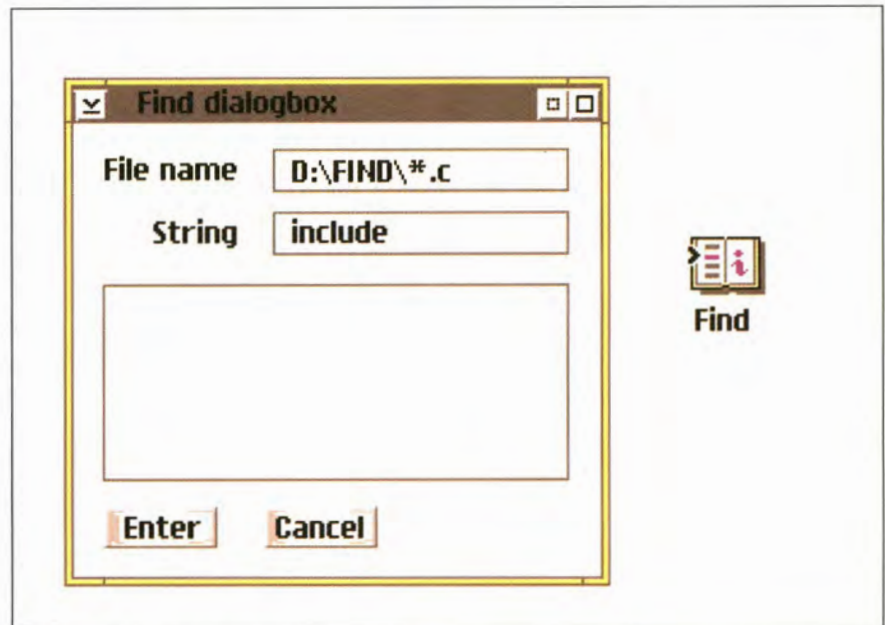


Figure 2. Dialogue of example program

```
#undef SOM_CurrentClass
#define SOM_CurrentClass SOMInstance
SOM_Scope void SOMLINK find_wpInitData(Find *somSelf)
{ FindMethodDebug("Find","find_wpInitData");
  parent_wpInitData(somSelf) }

SOM_Scope BOOL SOMLINK find_wpModifyPopupMenu
(Find *somSelf, HWND hwndMenu, HWND hwndCnr, ULONG iPosition)
{ FindMethodDebug("Find","find_wpModifyPopupMenu");
  return (parent_wpModifyPopupMenu(somSelf, hwndMenu,
  hwndCnr,iPosition)) }

SOM_Scope BOOL SOMLINK
find_wpMenuItemSelected(Find *somSelf, HWND hwndFrame, ULONG
MenuId)
{ FindMethodDebug("Find","find_wpMenuItemSelected");
  return(parent_wpMenuItemSelected(somSelf,hwndFrame,MenuId)) }

SOM_Scope ULONG SOMLINK find_wpDelete(Find *somSelf,
ULONG fConfirmations)
{ FindMethodDebug("Find","find_wpDelete");
  return (parent_wpDelete(somSelf,fConfirmations)) }
```

Figure 3. Results of compiling the `.CSC` and `.C` files



```
extern VOID    SOMLINK SOMInitModule( VOID );
extern int     InitializeFind(VOID);
extern BOOL    Findmain(Find *,int,char **);

int InitializeFind()
{CHAR    ErrorBuffer[100] ;

    if (DosLoadModule((PSZ) ErrorBuffer,
                      sizeof(ErrorBuffer),
                      "FIND", &hmodule) )
        return FALSE;
    FindIcon = WinLoadPointer( HWND_DESKTOP, hmodule, 101 );
    if (FindIcon==(HPOINTER)0)
        DebugBox("FAILURE","!!!load Icon failed");
    return TRUE }
```

Figure 4. Adding the business logic

```
if (FindIcon)
    _wpSetIcon(somSelf,FindIcon);
    argc  = 1;
    argv[0] ="find";

    Findmain(somSelf,1,argv);
```

Figure 5. Additions to find_wpInitData

```
_wpInsertPopupMenuItems(somSelf,      /*this cases adds to menu*/
                          hwndMenu,
                          iPosition,
                          hmodule,
                          RC_IDD_FINDMENU,
                          0);
```

Figure 6. Additions to find_wpModifyPopupMenu

into the FIND.C template file. It also compiles the Find.CSC and Find.C files, then compiles the completed Find.C file into the final DLL.

Results of Compiling the .CSC and .C Files. Once the compilation is complete, a subdirectory (created as a temporary storage area for the build process) is created, contain-

ing the constructed version of the file FIND.C. At this point, an error message appears, indicating that the FINDDLG.C and INITTERM.C files do not exist. Because the files have not yet been created, this message can be ignored. Figure 3 shows the additions to the end of the Find.C template.

Add the Business Logic. After the make file has constructed the FIND.C template, edit the new Find.C source and add the business logic, defining code for the override methods wpInitData, wpModifyPopupMenu, wpMenuItemSelected, and wpDelete. The resulting additions to the FIND.C source file are shown in Figure 4.

Add the information in Figure 5 to find_wpInitData, the information in Figure 6 to find_wpModifyPopupMenu, the information in Figure 7 to find_wpMenuItemSelected, and the information in Figure 8 to find_wpDelete.

Prepare the Business Logic:

FINDDLG.C. The FINDDLG.C source file contains the logical code for searching a file specification for a string. An example of this code is given in the IBM Technical Report listed in the References section.

Prepare Code to Initialize the DLL:

INITTERM.C. The initterm.c module performs the initialization for the Find class the first time its DLL is loaded. To access the resource file that contains the dialogue, it obtains the HMODULE for the DLL. The pertinent code excerpts are presented in Figure 9.

Prepare the Global Class Data: FIND-

DAT.H. Finddat.h contains all class data that will be global. In this case, it contains the module handle needed to load the dialogue and change the icon.

Create the Dialogue Box Resource

File: FIND.RC. The FIND.RC file is the resource file used by the Find class to present the dialogue to the user. To allow access to the Find function, create any dialogue. The result of one dialogue is shown in Figure 2.

Prepare the Definition File for the DLL:

FIND.DEF. The definition file used to create the FIND.DLL is shown in Figure 10.



FIRST SOLUTION: USING IPC AND SHARED MEMORY

By making a few changes to the previous example, you can create a program that uses interprocess communication to work around the shell single-process problem. This method creates an extremely small process, called a stub, that runs inside the Workplace Shell. The stub sends events to a second process operating completely outside of the Shell, which reduces the amount of code running inside the shell's single process, at the same time reducing the risk of the stub being adversely affected by another program running in the Workplace Shell.

We chose to implement the external process using a Presentation Manager object window because object windows support interprocess communication as an inherent part of their design. We can therefore implement interprocess communication with a minimal investment in time and

```
switch (MenuId) {
    case RC_MI_FIND:
        WinPostMsg(_Frame, WM_COMMAND,
            (MPARAM)MAKEULONG(FIND, FIND),
            (MPARAM)0L);

        break;
    default:
        return(parent_wpMenuItemSelected(somSelf, hwndFrame, MenuId)) }
return(TRUE);
```

Figure 7. Additions to find_wpMenuItemSelected

```
FindData *somThis = FindGetData(somSelf);
WinPostMsg(_Frame, WM_QUIT, (MPARAM)0, (MPARAM)0);
```

Figure 8. Additions to find_wpDelete

result in a DLL and an executable file; the DLL operates within the Workplace Shell and the executable file operates independent of the Workplace Shell process.

in the external process FINDDLG.EXE.

Changes to FIND.MAK. The FIND.MAK file is modified by removing the compilation and linking requirement of the FINDDLG.C module. The only other change is in the OBJS line, from which the finddlg.obj call is removed.

Changes to FIND.C. The FIND.C module is modified to allocate a shared piece of memory so the interprocess communication's object window handle can be communicated back to the Find class. The interprocess communication is started from the wpInitData method. The wpInitData procedure then executes the FINDDLG.EXE program, which is passed the parameter containing the name of the shared memory. The computer's speaker beeps when the external process is started and again, with a longer beep, when it terminates.

The changes to the FIND.C program are limited to the wpInitData portion. New parameters (char sharename[256], char Errorbuffer[60], char parms[256], RESULTCODES result, and FindData *somThis = FindGetData(som-

The best way to work around the shell single-process problem is to use interprocess communication between an object running inside the shell process and a process running completely outside it.

code and without designing fancy protocols.

This example uses a small piece of shared memory to pass a handle between the stub and the external process.

Source Files. There are minimal changes needed to the code in the existing program. These changes

Changes to FIND.CSC. For the first solution, the instance data statement of the FIND.CSC module is pruned and a new instance variable, HWND *Frame, is added to provide a handle to the object window that will provide us with interprocess communication. As a result, the instance data is located



CAN YOU BELIEVE IT? A LAN THAT SOLVES PROBLEMS.

Believe it.

This LAN will answer your questions.

This LAN will deliver network solutions.

This LAN will make your job easier.

This LAN is the original independent publication 100% dedicated to network professionals. By subscribing you give yourself access to

the latest product performance evaluations. No other LAN publication offers this kind of timely, objective material for local, wide, and global network managers and product buyers.

Let us simplify your life — for less than \$20 a year. It almost makes you forgive it for just being a magazine.



Believe It!

How can I possibly lose? If I become dissatisfied with LAN Magazine at any time, I may cancel my subscription and receive a *full refund* of my entire \$19.97 investment.

Check one:

- ☐ Payment enclosed.
- ☐ Bill me.

☐ Yes! Please start my full-year subscription (12 issues) to LAN Magazine for just \$19.97.

Name _____ Title _____

Company _____

Address _____ City _____

State/Province _____ Zip/Postal Code _____

Note: Subscription rate is for US delivery only. Canadian delivery: US\$25.97. International delivery: US\$34.97 for surface mail; US\$59.97 for airmail. Please send payment with order in US funds drawn on a US bank. Please allow 6-8 weeks for delivery. CA residents add \$1.45 state tax.

3 Easy Ways To Subscribe!

BY PHONE: 1-800-234-9573

BY FAX: 1-415-905-2233

BY MAIL:
LAN Magazine
P.O. Box 58123
Boulder, CO 80322-8123


```

#define INCL_DOSPROCESS
#include <stdlib.h>
#include <stdio.h>
#include <os2.h>

extern BOOL DLLInit( void );
extern BOOL DLLUninit( void );

#define DebugBox(title, text) \
    WinMessageBox(HWND_DESKTOP,HWND_DESKTOP, (PSZ) text , \
        (PSZ) title, 20, MB_OK MB_INFORMATION )

BOOL DLLInit()
{InitializeFind();
 return(TRUE);
}

/*-----
NAME: DLLUninit( VOID )

DESCRIPTION:
    This is the dynalink module exit routine.
    This routine will be called right before the module is being unloaded.
-----*/
BOOL DLLUninit()
{ return(TRUE);}

int _CRT_init(void);

int _CRT_term(unsigned long);

#pragma linkage( _DLL_InitTerm, system )

unsigned long _DLL_InitTerm(unsigned long modhandle, unsigned long
flag)
{ switch (flag)
    { case 0:
        if (_CRT_init() == -1)
            return 0UL;
        DLLInit();
        break;
        case 1:
            DLLUninit();
            _CRT_term(0UL);
            break }
    return 1UL }

APIRET DosGetThreadInfo(PTIB *pptib,PPIB *pppib)
{ return(DosGetInfoBlocks(pptib,pppib) ) }

```

Figure 9. Preparing the code to initialize the DLL: INITTERM.C



```
; FileName: find.def.
LIBRARY Find INITINSTANCE
DESCRIPTION 'Find Class Library - (c) Copyright IBM 1991'
PROTMODE
DATA MULTIPLE NONSHARED LOADONCALL
EXPORTS
    SOMInitModule
    FindCClassData
    FindClassData
    FindNewClass
    DialogProc
    ObjectWndProc
```

Figure 10. Preparing the definition file for the DLL: FIND.DEF

```
_Frame = (HWND *)0; /*initialize*/
sprintf(sharename, "\\SHAREMEM\\%lx.FND", (ULONG)somSelf);
DosAllocSharedMem((PPVOID)&_Frame,
    sharename, 4096, PAG_COMMIT | PAG_READ | PAG_WRITE);

memset(&parms, 0, sizeof(parms)); /*parms format*/
strcpy(parms, "find");
sprintf(parms+5, "%s", sharename);
DosExecPgm(Errorbuffer, sizeof(Errorbuffer),
    EXEC_ASYNC, parms, 0, &result, "FINDDL.G.EXE");
```

Figure 11. Changes to find_wpInitData

```
if (_Frame!=(HWND)0)
    WinPostMsg(*_Frame, WM_COMMAND,
        (MPARAM)MAKEULONG(FIND, FIND),
        (MPARAM)0L);

break;
```

Figure 12. Changes to find_wpMenuItemSelected

Self)) are added. The calls shown in Figure 11 replace Findmain(somSelf, argv) in find_wpInitData.

An object window is posted to communicate to an external process.

An additional check is then added in find_wpMenuItemSelected, and the pointer to _Frame replaces _Frame in the WinPostMsg call, resulting in the code shown in Figure 12.

The same changes are made in find_wpDelete, resulting in the call WinPostMsg(*_Frame, WM_QUIT, (MPARAM), (MPARAM)0) being activated when (_Frame!=(HWND)0) is true.

Changes to the FIND.DEF. The EXPORTS statement of the FIND.DEF file has been changed, as the window Procs are no longer contained in the object class. The new EXPORTS statements SOMInitModule, FindCClassData, FindClassData, and FindNewClass, are also needed.

New Make File for the External Process: FINDDL.G.MAK. A new make file must be created to build the FINDDL.G.EXE application used by the Find class. This creates the external process with which the find class is communicated. Changes to the standard make file for the existing program are shown in Figure 13.

Changes to FINDDL.G.C. The FINDDL.G.C module is modified to add a main body for a program; the threaded execution code has been removed, and changes to Finddlg.c are limited to the new parameters (HAB _hab, HMq _hmq, QMSG _qmsg, HWND _Frame). The parameter in the WinDlgBox call in ObjectWndProc is also changed from hmodule to 0L, and a new main procedure is created, as shown in Figure 14.

Definition file for the External Process: FINDDL.G.DEF. A definition file is required to build the FINDDL.G.EXE program. The file changes are shown in Figure 15.

Summary of the First Solution. The changes made to these source files provide a new Find program. With this program and an icon created in a file called FIND.ICO (not shown in this article), you can create a Workplace Shell object using SOM. Without the limits of the shell's single process, the object implements



```

CC      = icc /c /Ti /G3 /Ge /Gs /Kb
LINK    = link386
LDFLAGS= /co /noi /map /no1 /nod /exepack /packcode /packdata /align:16
LIBS    = os2386.lib dde4sbs.lib dde4nbs.lib

OBSJ    = finddlg.obj

.c.obj: $(CC) -I$(HPATH) -c $<

all: finddlg.exe

finddlg.obj: *.c *.h

finddlg.exe: *.def $(OBSJ) find.res $(LINK) $(LDFLAGS) $(OBSJ),$@,,
            $(LIBS),$*; rc find.res *.exe

find.res: find.rc rc -r find.rc find.res

#INCLUDES -C=ih

```

Figure 13. Changes to standard MAKE file

interprocess communication with a standard PM object window. This facilitates a standard message communication protocol. The named shared-memory allocation is used only to supply the handle to the

solution's source files FIND.CSC, FIND.C, and FINDDLG.C, shared memory need not be used. This example uses window enumeration to find the object window belonging to the external process.

inside the shell's process can easily locate and then communicate with the external process.

Changes to FIND.CSC. By initializing `_Frame` to `(HWND)0`, FIND.CSC is changed to create a window name to be passed to the external process. To ensure uniqueness, the name is based on the SOM ID for the object running inside the shell's process. In addition, inside `find_vpMenuItem-Selected` and `find_vpDelete`, if `(GetObjectWindow(somSelf))` replaces `if _Frame != (HWND)0`.

Changes to FIND.C. The FIND.C code is changed to create an object window using the prespecified name, as shown in Figure 16.

Changes to FINDDLG.C. The only change to FINDDLG.C is inside registering the class and creating the window, where the previous parameter (PSZ) "FindObjectClass" is replaced with (PSZ) `argv[1]`.

Without the limits of the shell's single process, the object implements interprocess communication with a standard PM object window.

object window. If data exchange is required, `WM_DDE*` messages should be used, rather than interprocess communication messages.

SECOND SOLUTION: USING THE SOM ID

With a few changes to the first

This solution can be accomplished by passing a parameter containing a specified name to FINDDLG.EXE at execution. FINDDLG.EXE can then use the specified name when it creates an object window. Because the name has been prespecified, the `Find` class running



```
int main(int argc, char **argv)
{HWND *shared;
 int rc;

    DosBeep(100,100);          /*Indicate that it has been started*/

    if (argc<2) exit(-1);

    if (rc=DosGetNamedSharedMem((PPVOID)&shared,
                                argv[1] , PAG_READ | PAG_WRITE))
        exit(-1);

    _hab = WinInitialize((ULONG)0L);
    _hmq = WinCreateMsgQueue(_hab, (LONG)0L);
    WinRegisterClass(_hab, (PSZ)"FindObjectClass",
                    (PFNWP)ObjectWndProc, (ULONG)0L, (ULONG)4L);
    *shared=_Frame = WinCreateWindow(HWND_OBJECT,
        (PSZ) "FindObjectClass",
        (PSZ) "Object Window", 0L,
        (ULONG)0L,(ULONG)0L,(ULONG)0L,(ULONG)0L,
        (HWND)0, HWND_TOP, 101,
        (ULONG)0, (PVOID)0);
    while (WinGetMsg(_hab, (PQMSG)&_qmsg,
        (HWND)NULL, (ULONG)0L, (ULONG)0L ) ){
        if (_qmsg.msg==WM_QUIT) break;
        WinDispatchMsg( _hab, (PQMSG)&_qmsg ) }
    DosBeep(1000,1000);          /*indicate that it has ended*/
    WinDestroyWindow( _Frame );
    WinDestroyMsgQueue( _hmq );
    WinTerminate( _hab ) }
```

Figure 14. Changes to FINDDLG.C

SUMMARY OF THE SECOND SOLUTION

With a few additional changes, it is possible to alleviate the need for shared memory to be used to communicate the name of the external process' object window. Passing the external process a unique name is an even cleaner solution. We recommend either the first or second solution, which avoid the problems associated with the Workplace Shell single process.

NAME	FINDDLG WINDOWAPI
STUB	'OS2STUB.EXE'
CODE	MOVEABLE
DATA	MOVEABLE MULTIPLE
HEAPSIZE	8192
STACKSIZE	8192
EXPORTS	DialogProc
	ObjectWndProc

Figure 15. Changes to FINDDLG.DEF



Richard Redpath, IBM Corp., 11000 Regency Pkwy., Cary, N.C. 27512. Redpath is an advisory software engineer in the IBM office platform tools architecture group. He began working for IBM in 1984; before that, he worked for the Carnegie-Mellon Robotics Institute in autonomous mobile robot programming and for the T.J. Watson Research Center's advanced workstation group.

Joe Coulombe, IBM Corp., 11000 Regency Pkwy., Cary, N.C. 27512. Coulombe is an object and user interface architect in the IBM programming systems group. Previously, he was a key architect of the 1991 CUA™ user interface architecture. Coulombe joined IBM in 1989 with extensive experience in software design and development and multimedia design and production.

Susan Henshaw, IBM Corp., 11000 Regency Pkwy., Cary, N.C. 27512. Henshaw is an object and user interface designer in the IBM programming systems group. Since 1989, she has been a key member of the Advanced CUA user interface design team as an architect, designer, prototyper, and consultant.

REFERENCES

OS/2 2.0 Volume 4 Application Development (IBM Doc. GG24-3774-00)

Redpath, Richard, and Joe Coulombe. "An Interim Solution to the Single-Process Problem of OS/2 2.0," IBM Technical Report TR29.1427.

```
BOOL GetObjectWindow(Find *somSelf)
{
    HENUM henum;
    HWND hwnd;
    CHAR name[256];
    FindData *somThis = FindGetData(somSelf);

    if (_Frame!=(HWND)0) return(TRUE);
    henum = WinBeginEnumWindows(HWND_OBJECT);
    while (hwnd= WinGetNextWindow(henum)) {
        WinQueryClassName(hwnd, sizeof(name), name);
        if (strcmp(name, _instancename)==0)
        {
            _Frame=hwnd;
            WinEndEnumWindows(henum);
            return(TRUE) } }
    WinEndEnumWindows(henum);
    return(FALSE) }

```

Figure 16. Changes to FIND.C



OS/2 PRODUCTS & SERVICES

Consulting **OS/2 PM CUA**

Management	Programming
Analysis	Testing
Design	Installation
Real-Time Controls and Device Drivers	Maintenance

Real-Time Technologies
1001 Green Bay Rd. Winnetka, IL 60093
(708) 380-3584

Schedule Programs and Reminders ...



Chron v3.0

Event Scheduler for OS/2

Only \$69



Hilbert Computing
1022 N. Cooper
Olathe, KS 66061

Voice: (913) 780-5051
BBS/Fax: (913) 829-2450
CIS: 73457,365

CPPCOMM Version 3.0

Multi-threaded Communications Library for OS/2 2.0 and Windows NT with Full Source Code

Also includes DOS and 16-bit Windows versions. Fully object-oriented code, 70 pp. documentation. C++ class objects encapsulate UART and API so that code is seamlessly portable between DOS, Windows, OS/2 and NT operating systems. Includes high performance X/Y/Zmodem protocols and many other features! IBM, Microsoft, Borland, Zortech compilers. \$79 postpaid. **Lookout Mountain Software, 611 Alma Ave., Pueblo, Colorado 81004-1607; Data/Fax (719) 545-8572.**

Attention Mathematica users:
Everything you need to know to get the most out of your math programming is in

THE MATHEMATICA JOURNAL

Call or fax today for information about subscriptions and special issues:
Phone orders: 415-905-2332
Fax orders: 415-905-2233

NEURAL NETWORK

SPECIAL REPORT

Your comprehensive guide to neural network tools and the companies that make them - all new for 1992!

All for only \$7.95.

To order, call 415-905-2376, or fax to 415-905-2233 now

TCP/2™ tcp/ip for OS/2

"TCP/2™ is, by far, the best implementation of the tcp/ip application suite for DOS and OS/2 available from any source"

Microsoft ITIS

Essex Systems, Inc.

One Central Street Phone (508) 750-6200
Middleton, MA 01949 Fax (508) 750-4699



Custom Entryfields allow formatting for fields like Phone Number, Dates, Social Security Number, Drivers License, etc.

Easy to use. Just change the window class in your resource file.

ENTRYFIELD PICTURE MASKS for OS/2

Download free demo of Impact Entryfields from our BBS.
(818) 879-7405

Add Impact to your application



IMPACT SOFTWARE

To order call or write
(800) 676-9390
(818) 879-5592
FAX (818) 879-5593

5889 Kanan Rd.
Suite 330
Agoura Hills, CA 91301

SUBSCRIBE TODAY!

Only \$39.95 for a full year subscription to the premier source of tools and techniques for the OS/2 software developer: **OS/2 DEVELOPER**

Call today and don't miss a single issue:

1-800-WANT-OS2



This article builds on "Demystifying Custom Controls" (Winter '93), explaining in more detail issues confronting the design and implementation of a custom control. **BY MARK A. BENGE and MATT SMITH**

Designing Custom Controls



Mark Benge



Matt Smith

IN THE LAST ISSUE, WE PRESENTED A graphical button that contained image and text components. We quickly explained some of the design and implementation issues that concerned the control. In this installment of the article, we will explain in more detail the issues confronting the design and implementation of a custom control.

FIRST PRINCIPLES

Most new controls are a clean slate, or "tabula rasa," when they are first conceived. A control is just another window. It can be thought of as a device for displaying information for, and gathering input from, users. Custom controls are not that hard to create as long as you have the proper information. Figure 1 shows four simple but useful controls not presently found in the OS/2 Presentation Manager.

The 3-D separator, the simplest control in Figure 1, uses the `GpiLine` function to create shadow and highlight lines. Figure 2 shows the source code that implements painting the control. (This and other sample source code fragments shown in this article can be found on CompuServe in LIB13 of the OS2DF2 forum, or through other electronic means outlined at the end of the article.)

The first component of the separator, the upper line, is drawn in the shadow color, while the second component, the lower line, is drawn in white, causing the three-dimen-

sional effect. There's not much to it, especially since the control does not interact with the user through either the keyboard or the mouse. The sample code provides for two styles, vertical and horizontal. Through other processing in the control, the rectangle coordinates of the control are determined and that value is used to determine the starting and ending points of the lines.

The pattern control (the halftone area in Figure 1) uses the `GpiPattern` function along with the draw and fill capabilities of the `GpiBox` function, as shown in Figure 3. Again, the painting of the control is not complex. The sample control allows for each of the different `Gpi` pattern types as a style. The painting routine determines the pattern style by querying the control style with `WinQueryWindowULong` with an index of `QWL_STYLE` and masking out the styles that are not valid `PATSYM_*` values.



Figure 1. Example of custom control

The 3-D frame control, the third control from the left in Figure 1, uses the principles demonstrated in the 3-D line control. `GpiBox` is used to draw the two boxes, first the light box and then the shadow box. The boxes are offset from each other by one pixel, giving a



```
/* Paint the Control */
case WM_PAINT :

    /* Get the address of the text from */
    /* the control's reserved memory */

    plf = (PLINEFIELD)WinQueryWindowULong(hWnd,
                                           QWL_USER);

    hPS = WinBeginPaint(hWnd, (HPS)NULL,
                       (PRECTL)NULL)
    GpiSetColor(hPS, SYSCLR_SHADOW);

    GpiMove(hPS, plf->aptl);
    GpiLine(hPS, &plf->aptl[1]);

    GpiSetColor(hPS, CLR_WHITE);

    GpiMove(hPS, &plf->aptl[2]);
    GpiLine(hPS, &plf->aptl[3]);

    WinEndPaint(hPS);
    break;
```

Figure 2. 3-D separator control painting code fragment

```
/* Paint the Control */
case WM_PAINT :

    /* Get the address of the control info from */
    /* the control's reserved memory */

    ppat = (PPATTERN)WinQueryWindowULong(hWnd,
                                           QWL_USER);

    /* Get the presentation space for the window */
    /* and draw the grid which the buttons are */
    /* placed */

    hPS = WinBeginPaint(hWnd, (HPS)NULL, (PRECTL)NULL)
    GpiSetPattern(hPS,
                  (LONG)WinQueryWindowULong(hWnd,
                                              QWL_STYLE) &
                  0x5f);

    ptl.x = ptl.y = 0L;
    GpiMove(hPS, &ptl);
    ptl.x = ppat->rcl.xRight;
    ptl.y = ppat->rcl.yTop;
    GpiBox(hPS, DRO_FILL, &ptl, 0L, 0L);
    /* Release the presentation space */
    WinEndPaint(hPS);
    break;
```

Figure 3. Pattern control painting code fragment

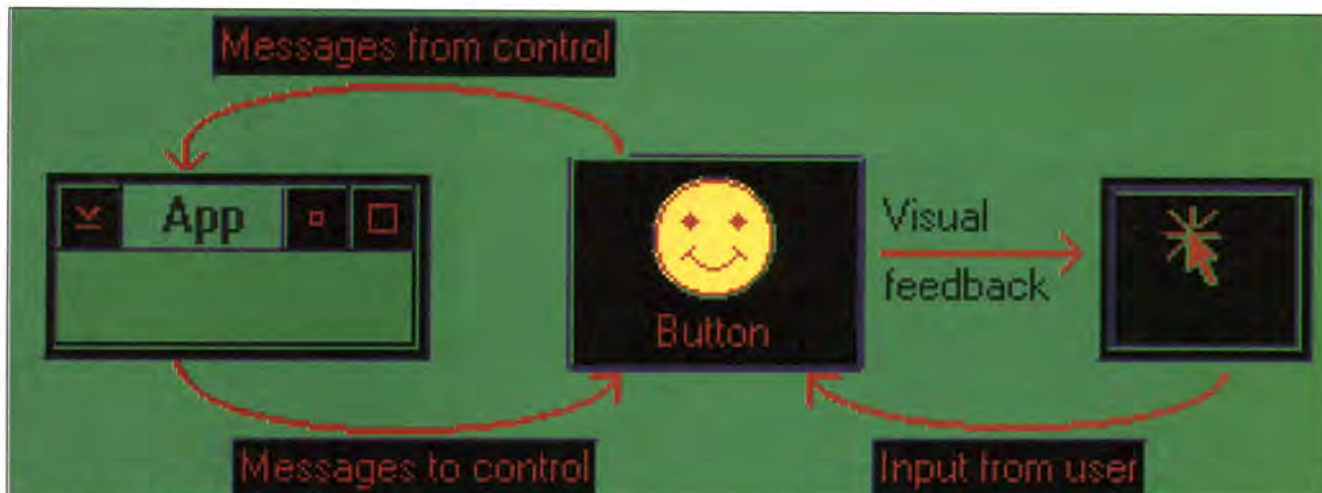


Figure 4. Control interaction diagram

three-dimensional effect.

Finally, the 3-D text, the last control in Figure 1, is a composite control consisting of a static text window and an outer line. Any special presentation parameter processing is done when the text window is created; this ensures that the text is

the handling of messages; most messages of interest, such as `WM_PRESPARAMCHANGED`, need to be passed to the text window with `WinSendMessage`.

The architecture of the source code of the example controls is similar to that of the others. Its architecture is the foundation of a base control (which, in Presentation Manager terms, begins with the basic static control type).

CONTROL TYPOLOGY

The first of the basic control types is the static control, used basically to display information. Static controls found in Presentation Manager include text, group boxes, frames, icons, and bitmaps. These only display information; they allow for user input through neither the keyboard nor the mouse pointer.

Static controls are the easiest to implement; because they are concerned only with displaying information, they don't need a well-defined communication mechanism between the control and its owner.

The second type of control, however, must consider input. Through the mouse or keyboard, users can manipulate a control to inform the owning application of a specific selection. Examples of this type of

control are the button, slider, scroll bar and value set controls of Presentation Manager, which display only visual information such as scale, position, color, and so on. Users make a selection decision based upon the visual information presented by the controls.

A vertical scroll bar, for example, consists of a scroll button, shaft, and elevator or thumb. When the thumb is just below the top scroll button, the top scroll button is disabled, indicating to the user that the only direction of movement is down. Users can either click the mouse pointer on the down scroll button, click the mouse on the shaft below the thumb, or select the thumb and drag it downward toward the bottom scroll button. This type of operation helps users understand certain conditions within the application and allows the user to inform the application of a desire to change from a current condition to a new one, a structure that can be thought of in terms of action and reaction.

The third type of control, a combination of the first two types, must extend input to allow for selections; information displayed can be visually scanned and selected. The list-box, combination box, and spin button are good examples of this type of control.

Understanding the role of a control allows you to determine how it should operate with users and the owning application.

in the color and font required. The 3-D frame is then drawn around the text window using a technique similar to that described for the 3-D frame. This time, the left and top edge are drawn in the shadow color with `GpiPolyLine`; the color is then changed to white before `GpiPolyLine` is used to draw the right and bottom edges. This causes the text to appear as though within a sunken area.

Special consideration is given to



Message	Buttons	Container	Combo Box	Entry Field	List Box	Note Book	Slider	Spin Button	Static	Value Set
WM_ADJUSTWINDOWPOS			•	•	•			•	•	
WM_BUTTON1DBLCLICK	•	•	•	•	•	•	•	•		•
WM_BUTTON1DOWN	•	•	•	•	•	•	•	•		•
WM_BUTTON1UP	•	•	•	•	•	•	•	•		•
WM_BUTTON2DBLCLICK				•			•			
WM_BUTTON2DOWN		•		•	•		•			•
WM_BUTTON2UP		•					•			•
WM_BUTTON3DBLCLICK				•						
WM_BUTTON3DOWN		•		•	•					
WM_CHAR	•	•	•	•	•	•	•	•		•
WM_CLOSE						•				
WM_CONTROL	•	•	•	•	•	•	•	•		•
WM_CONTROLPOINTER		•	•			•	•			•
WM_CREATE	•	•	•	•	•	•	•	•	•	•
WM_DESTROY	•	•	•	•	•	•	•	•	•	•
WM_DRAWITEM		•				•	•		•	
WM_ENABLE	•		•	•	•		•	•		•
WM_ERASEBACKGROUND		•								
WM_HELP		•				•				•
WM_HITTEST			•						•	
WM_HSCROLL		•			•					
WM_MATCHMNEMONIC	•									
WM_MOUSEFIRST				•	•					
WM_MOUSELAST				•	•					
WM_MOUSEMOVE	•	•	•	•	•	•	•		•	•
WM_MOVE			•							
WM_PAINT	•	•	•	•	•	•	•	•	•	•
WM_PRESPARAMCHANGED	•	•	•	•	•	•	•		•	•
WM_QUERYCONVERTPOS	•	•		•	•			•	•	
WM_QUERYDLGCODE	•			•					•	
WM_QUERYWINDOWPARAMS	•		•	•			•		•	•
WM_SETFOCUS	•	•		•	•	•	•	•	•	•
WM_SETSELECTION			•	•				•		
WM_SETWINDOWPARAMS	•		•	•			•		•	•
WM_SIZE		•					•	•		
WM_SYSCOLORCHANGED	•			•	•				•	•
WM_SYSVALUECHANGED	•			•						
WM_TIMER		•	•	•	•		•	•		
WM_TRANSLATEACCEL						•				
WM_VSCROLL		•			•					
WM_WINDOWPOSCHANGED	•		•	•	•				•	

Table 1. Messages handled by Presentation Manager control



Message	Use
BM_CLICK	Allows the owning application to simulate the clicking of pushbutton
BM_QUERYCHECK	Used to determine button check state
BM_QUERYCHECKINDEX	Used to determine the index of the button checked within a group of buttons
BM_QUERYHILITE	Used to determine if a button is highlighted
BM_SETCHECK	Sets the check state of a button
BM_SETDEFAULT	Sets the button state to its default condition
BM_SETHILITE	Sets the highlight state of the button

Table 2. Button-specific messages

```
typedef struct _BTNCDATA /* btncd */
{
    USHORT cb; /* Structure Size */
    USHORT fsCheckState; /* Button Check State */
    USHORT fsHiliteState; /* Button HighLite State */
    LHANDLE hImage; /* Icon/Bitmap Handle */
} BTNCDATA;
```

Figure 5. Button CTLDATA structure

Message	Use
BN_CLICKED	Button clicked on
BN_DBLCLICKED	Button has been doubleclicked on
BN_PAINT	User drawn button requires painting

Table 3. Button notification messages

The listbox control within Presentation Manager is comprised of a list area and scroll bar. The scroll bar operates like the one described previously. The list area operates as a group of static text controls. When combined, the actions of users on the scroll bar determine the contents of the list area. As users cause the scroll bar to change position, through one of several

means (scroll button, page up and down, dragging), the list area is scrolled in conjunction with it. A further interaction allows users to select an item within the list area, which usually indicates to the application that a user wants to manipulate or use that item.

The final type of control uses all the techniques of the others, plus allowing editing of displayed infor-

mation. Examples of this type are the Presentation Manager entry field and the multiple-line entry field. Here, the control displays information and allows it to be selected and edited.

A control must be able to:

- Display some type of information
- Accept input if it is to denote a selection or manipulation
- Allow the entering or editing of information.

Understanding the role of a control allows you to determine how it should operate with users and the owning application. The control interacts with users both visually and through the keyboard and mouse, allowing interaction with the application through various messages. The application may also communicate with the control through messages. Figure 4 shows this interaction process.

In all cases, the control displays some information. As shown in Figure 4, the control acts in a sort of isolation: it is an entity that interacts through a defined set of conventions with the application and the user. The control must be designed to assume that it cannot depend on the application for necessities such as data in its life-span.

The control must be created to work from the desktop as though no application owned it. Although this idea may seem strange, it allows controls to work under almost any condition. It is possible for the control to be created with the desktop as the parent and owner. This means that the communication flow for control-specific messages is only from the control to the desktop. The messages most likely won't be understood by the desktop, and the desktop will not send any control specific messages to the control, because it has no



Powerful New Tools for OS/2 Programming

The WorkFrame/2 product ... because the best environment for application development is the one you create yourself.

With WorkFrame/2, you can integrate your choice of development tools – including those for DOS and Windows. It's open, configurable and language independent. And it's easy to customize the WorkFrame/2 interface to create your own development environment.

The concept is simple. WorkFrame/2 organizes files into logical units called projects. By associating each project with your personal choice of compiler/debugger/linker/editor you can get the greatest productivity possible from all your development tools.

The C Set/2 product ... because application development should be fast – and simple!

C Set/2 delivers a one-two punch to help you create some of the fastest-performing OS/2-based applications possible.

First, the 32-bit C compiler enables your applications to exploit the speed and power of 386- and 486-based computers. It's the best high-performance code optimizer in the business. With the C Set/2 compiler, unsafe optimizations simply don't exist.

Second, C Set/2 comes with a fully interactive, full-function, source-level 32-bit Presentation Manager debugger. Just point your mouse and shoot, using the graphical PM user interface – or use the keyboard.

Either way it's easy. And you'll get instant feedback on the screen to verify what you're doing. Debugging has never been so simple!

And there's more. The C Set/2 compiler conforms to industry standards – including ANSI C and ISO/IEC – and offers Microsoft C compatibility. With features like a full suite of run-time libraries and 32-16 bit linkage, you can be sure C Set/2 will provide the function and flexibility you need to make application development fast and simple – the way it should be.

OS/2 2.0 Developer's Toolkit ... because it takes the right set of tools to build powerful applications.

OS/2 2.0 Developer's Toolkit is the perfect companion to use with C Set/2. It contains a variety of language-independent application build and productivity tools. For the C Set/2 compiler, Toolkit provides the system linker and system header files. It also contains the import libraries and the NMAKE utility you need to dramatically increase the capabilities of C Set/2 to build powerful applications.

To order or get more information on how IBM application development tools can work in your OS/2 environment,

in the USA call 1-800-342-6672
in Canada call 1-800-465-7999

Making good things happen in application development... 


```
WinCreateWindow(HWND hwndParent, PSZ pszClass,
                PSZ pszName, ULONG flStyle, LONG x,
                LONG y, LONG cx, LONG cy,
                HWND hwndOwner, HWND hwndInsertBehind,
                ULONG id, PVOID pCtldata,
                PVOID pPresParams);
```

Figure 6. WinCreateWindow definition

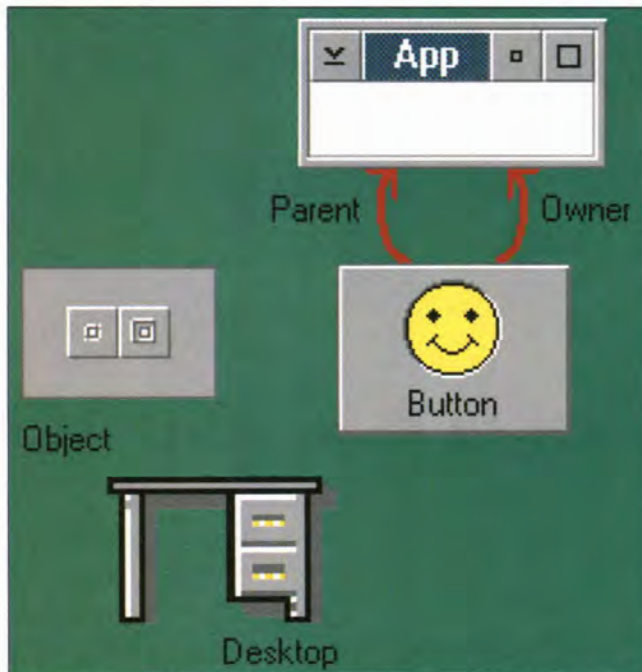


Figure 7. Application parent-owner-control relationships

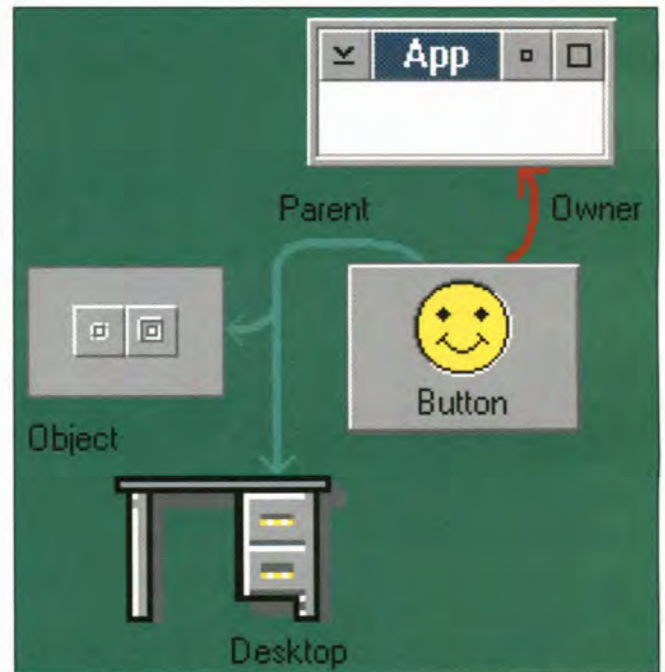


Figure 9. Desktop/object window parent-owner-control relationships

understanding of the control's message conventions.

Controls should be designed to get all required resources during creation and perform as much setup as possible without further messages from their creator. Generally, this is done through the CTL-
DATA mechanism, part of the WinCreateWindow function shown in Figure 6 (parameter 12, P-
VOID pCtldata). While this use of CTL-
DATA may be sufficient during creation, care should be taken so if the data structure is not present, messages to the control with the appropriate resources can achieve the same effect, still allowing the control to operate even in a minimal state.

While designing the control, continually remind yourself how to maintain its responsiveness. You

can do this through prudent coding of sections within the control, perhaps with the use of threads. Delays in responsiveness will cause the control to respond unevenly, somewhat like the notebook control in OS/2 2.0.

WHAT'S IN A DESIGN?

A control can be as simple as a line or pattern and as complex as the container control. It need not be a unique entity, but can be a composite control, made up of other controls, similar to the 3-D text control presented earlier.

You have to plan and architect a control's appearance and how it is to interact with the application and with users. This includes the external data (CTL-
DATA or messages) passed to the control through stan-

dard functions like WinSetWindowText or WinSetDlgItemText, as well as special-purpose messages intended for use only by the control.

MESSAGES AND MORE MESSAGES

One of the major design issues to contend with is the handling of messages sent by Presentation Manager to your control. Unfortunately, very little in the OS/2 Technical Library mentions the creation and handling of custom controls. Table 1 outlines the messages handled by the various Presentation Manager controls; you can guide the message handling design of your control by comparing your control's characteristics with those of the Presentation Manager controls.

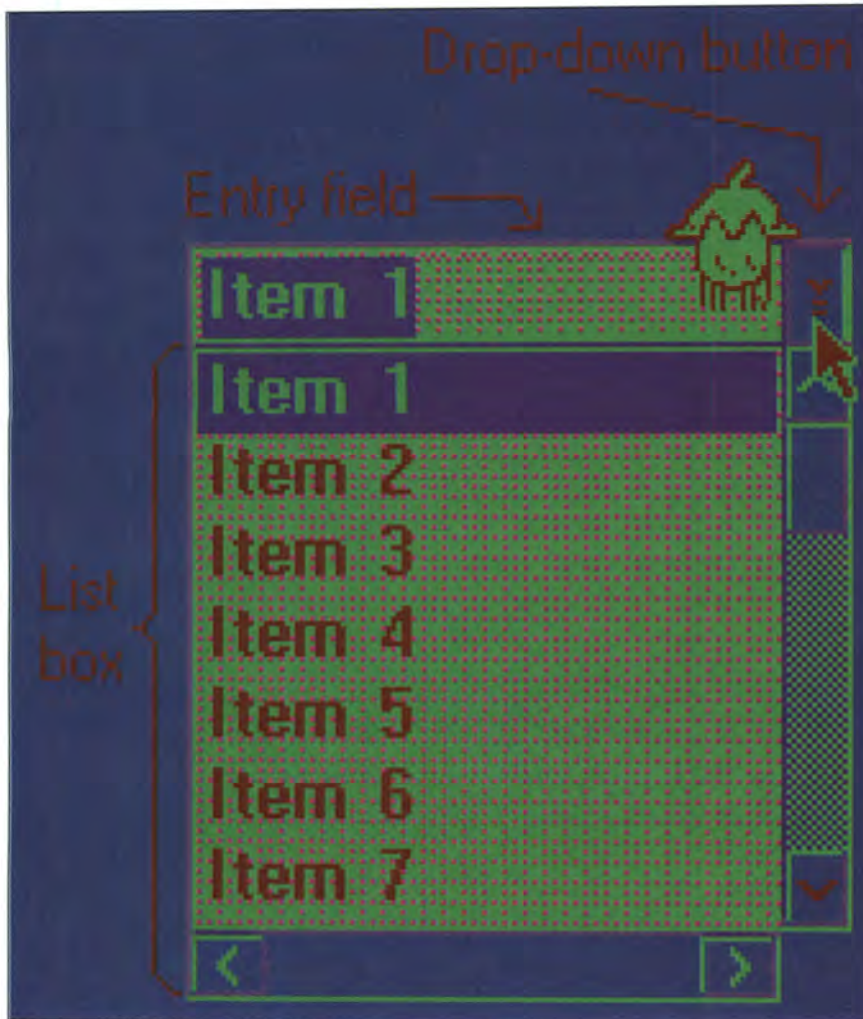


Figure 8. Combo box

```
typedef struct _CREATESTRUCT /* crst */
{
    PVOID pPresParams; /* Pres. Parameters Pointer */
    PVOID pCtlData; /* Control Data Pointer */
    ULONG id; /* ID Value */
    HWND hwndInsertBehind; /* Z-Order Placement */
    HWND hwndOwner; /* Owner Window Handle */
    LONG cy; /* Window Height */
    LONG cx; /* Window Width */
    LONG y; /* Window Vert. Placement */
    LONG x; /* Window Horz. Placement */
    ULONG flStyle; /* Window Style */
    PSZ pszText; /* Window Text */
    PSZ pszClass; /* Class */
    HWND hwndParent; /* Parent Window Handle */
} CREATESTRUCT;
```

Figure 10. CREATESTRUCT structure

It is recommended that you be prepared to handle all messages received by your control. In reality, however, you should handle only those messages that will affect your control. For example, you should handle the `WM_PAINT` message, whereas you may need to handle the `WM_SETSELECTION` message. Finally, if you want to create a composite control, you may have to handle those messages in the subclass procedure of a control that you are using to create your control.

After deciding which messages you wish to handle that will be specifically sent to the control by Presentation Manager, you need to define messages sent by the owning application. These messages are control-defined and unique to the control. Table 2 shows the list of such messages defined for Presentation Manager buttons.

Message Use. These messages should be coordinated with the `CTL_DATA` structure where possible. Figure 5 shows the corresponding Presentation Manager button `CTL_DATA` type structure, `BTNCDATA`.

The message `BM_SETCHECK` corresponds to the `fsCheckState` field, while the `BM_SETHILITE` message corresponds to the `fsHiliteState` field.

In conjunction with the `WM_CONTROL` message are the notification messages that the control uses to notify the owning application of a significant event. Table 3 shows the list of notification messages used by Presentation Manager buttons.

THE FAMILY TREE

To understand the relationship between the family tree and the control, you need to understand how the control is ultimately created. All windows are eventually created by `WinCreateWindow`, defined in Figure 6. The first parameter is the parent window handle and the ninth parameter is the owner window handle. Usually, the parent and the owner are the same for



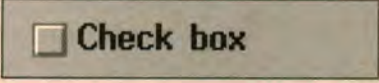
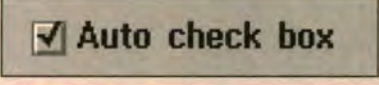
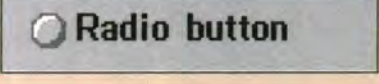
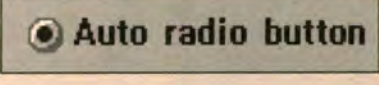
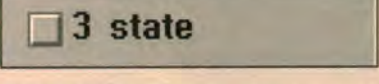
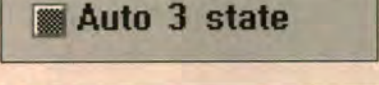
Base Style	Button
BS_PUSHBUTTON	
BS_CHECKBOX	
BS_AUTOCHECKBOX	
BS_RADIOBUTTON	
BS_AUTORADIOBUTTON	
BS_3STATE	
BS_AUTO3STATE	
BS_USERBUTTON	

Table 4. Button base styles

```
typedef struct _USERBUTTON /* ubtn */
{
    HWND    hwnd;          /* Button Window Handle */
    HPS     hps;           /* Presentation Space Handle */
    ULONG    fsState;       /* Current State */
    ULONG    fsStateOld;    /* Previous State */
} USERBUTTON;
```

Figure 11. USERBUTTON structure

controls, as shown in Figure 7. However, depending on how the control works, the owner and the parent could be different, as shown in Figure 9.

You can specify the owner and parent to be the same through the `WinCreateWindow` function; in this case, messages are sent to the owner window and the control is

displayed when the parent is displayed.

You can also use `WinCreateWindow` to specify differing parents and owners for the control. A good example of this is the combo box, shown in Figure 8, in which a composite control is comprised of three components: an entry field, a list box, and a drop-down button.

When the control is first created (assuming the style is drop-down or drop-down list), the entry field's owner and parent are the drop-down button. The list box used to display the drop-down list is created with the owner the drop-down button and the parent `HWND_OBJECT`. This relationship is shown in Figure 9.

This initially causes the list box component to be hidden, because descendants of object windows are not visible. Messages (`LM_*`) sent to the combo box handle are redirected to the listbox.

When a user clicks the mouse pointer on the drop-down button, the combo box changes the parent of the list box from the object window to the desktop, which allows it to be displayed. When a user selects an item within the list box, the notification message is sent to the drop-down button that owns it, not to the desktop (which is its parent). The message is then repackaged and sent on to the application that owns the combo box.

When you send a message back to the owning window, you send it to the owner handle. The owning application can set the parent to an object window while you are still processing information; in this case, the owner would still be an application window that receives all messages from the control. When the drop down button is pressed again, the list box can be hidden by changing the parent from the desktop back to the object window.

LIFE OF A CONTROL: AN EXAMPLE

The best way to understand control design is to dissect an existing control. We will now look at the Presentation Manager button control and explain what is happening under the covers. This will help you both understand the processing of existing Presentation Manager controls and model your control on them.



Additional Style Use

- **BS_AUTOSIZE** Button is sized to contents to ensure that contents are visible.
- **BS_BITMAP** Button will display graphic bitmap image supplied through text (#id) or through hImage of BTNCDATA structure. (This support was not implemented in OS/2 2.0 GA, but was implemented in the SP and OS/2 2.1.)
- **BS_DEFAULT** Push button is drawn with heavier border indicating button will automatically be selected if the ENTER key is pressed.
- **BS_HELP** Push button will post a WM_HELP message instead of the normal WM_COMMAND.
- **BS_ICON** Button will display graphic icon image supplied through text (#id) or through hImage of BTNCDATA structure.
- **BS_NOBORDER** Push button will be displayed with no border.
- **BS_NOCURSORSELECT** Auto radio button will not select itself when given the focus through the keyboard.
- **BS_NOPOINTINTERFOCUS** Focus will not be received by the button though the button is still selectable. This is generally used with the help pushbutton so that the currently focused control will be used to look up help from the window or dialogue subtable .
- **BS_SYSCOMMAND** Push button posts a WM_SYSCOMMAND instead of the normal WM_COMMAND message.

Table 5. Additional style use

WM_CREATE. Every control has three stages of life: a beginning, a middle, and an end. WM_CREATE, the beginning of a control's life, is received when the control is created through WinCreateWindow.

Two structures are presented through the mp1 and mp2 message parameters. The first structure, passed in the mp1 message parameter, corresponds to the PVOID pCtlData parameter of WinCreateWindow. The second structure, passed in the mp2 message parameter, contains a pointer to the creation structure shown in Figure 10. This structure helps you determine the parent and owner of the control, along with its internal ID value, style, text, position, and size. You will want to save most of this informa-

tion internally within a private data structure. You should save the data structure pointer within the window words of the control, which you have reserved with the WinRegisterClass function. The data is assembled from the parameters of WinCreateWindow. The elements of the CREATESTRUCT structure map to the parameters from right to left, as in the prototype in Figure 6.

The Presentation Manager button control determines if the BTNCDATA structure, shown in Figure 5, has been passed to it by checking if mp1 is NULL or contains a valid pointer. It then verifies that the structure is valid by checking its size, which is the first word of the structure. Thus, the first word or long value of the CTLDATA type

structure must contain its size. This value is used by OS/2 Presentation Manager to allocate memory for the structure before it is passed to the control.

Any relevant values within the BTNCDATA structure are recorded internally within the control's private data area to allow the correct representation of the control. General setup, such as transferring a copy of the text of the button to internal memory, is then performed.

If anything is incorrect, the control returns TRUE to indicate to the window manager that the creation of the control should stop. This may cause a domino effect depending on who is creating the control. If the control is being created by the



Dialogue Code	Meaning
DLGC_RADIOBUTTON	Identifies button as a radio button
DLGC_DEFAULT	Identifies button as default push button
DLGC_PUSHBUTTON	Identifies button as a non- default push button
DLGC_CHECKBOX	Identifies button as a check box

Table 6. Dialogue codes

typedef struct _WNDPARAMS	/* wprm	*/
{		
ULONG	fsStatus;	/* Query/Set Flag
ULONG	cchText;	/* Text Size
PSZ	pszText;	/* Text Pointer
ULONG	cbPresParams;	/* Pres. Parameters Size
PVOID	pPresParams;	/* Pres. Parameters Pointer
ULONG	cbCtlData;	/* Control Data Size
PVOID	pCtlData;	/* Control Data Pointer
}	WNDPARAMS;	

Figure 12. WNDPARAMS structure

dialogue manager through a call to `WinCreateDlg`, `WinDlgBox`, or `WinLoadDlg`, the entire dialogue creation will fail and the dialogue will not be displayed. Unfortunately, this behavior is not very well documented.

and the memory for private data is released back to the system. Also, any icons or bitmaps loaded by the control as part of the control's text string would be destroyed here.

WM_PAINT. When the button needs to show itself to the world, it occurs during the processing of the `WM_PAINT` message. The control determines the current style by checking through `WinQueryWindowU-Long` with the `QWL_STYLE` index to ensure that the control is painted correctly. The styles shown in Tables 4 and 5 are used as the basis for most painting. Using this information, the button then checks to see if it is visible; if it isn't, the rest of the processing is ignored, since it is unnecessary.

The button is painted using the bitmap, line, and text graphics calls. The bitmap calls are used to draw the radio button and check box components—unless it is a push

button, in which case the outline edges are drawn with the line functions. The text for the button is also drawn, including the mnemonic if one has been provided.

For the user button style, the button fills in a `USERBUTTON` structure, shown in Figure 11, that allows the owner of the control to paint the button in a manner specific to the owning application. A `WM_CONTROL` message is packaged with the `BN_PAINT` notification and is sent to the owner of the button.

WM_WINDOWPOSCHANGED. When the button changes position or size, this message is received and used to force a recalculation of the button dimensions.

WM_SETFOCUS. When the button receives the focus, the mouse pointer is setup for use by the button. This setup involves the retrieval of the default arrow shape. When the

Every control has three stages of life, a beginning, a middle, and an end.

With the normal return value of `FALSE`, the creation process will continue.

WM_DESTROY. The button control ends its existence when it receives the `WM_DESTROY` message. Usually, any clean up is performed here,

focus is being lost, it discards the mouse pointer.

WM_MOUSEMOVE. When using the mouse pointer setup through the `WM_SETFOCUS` message, the mouse pointer is displayed over the control after the button has sent a `WM_CONTROLPOINTER` message to the owning application, giving it a chance to change the pointer shape.

When the mouse is in capture mode after a `button 1 down` event, the mouse position is monitored to see if it is within or outside the button limits. When it is within the limits, the button is displayed in the highlighted state, and when it is outside the limits, the button highlight state is removed.

WM_BUTTON1UP. The button control ignores this message if the mouse capture is not active. When capture is active, it is removed and a `BN_CLICKED` notification message is packaged for the `WM_CONTROL` message that is sent back to the owner of the button. However, if it was a double click, `BN_DBLCLICKED` is packaged for the `WM_CONTROL` message.

WM_BUTTON1DBLCLICK. The event is recorded to allow the `BN_DBLCLICKED` notification to be sent after the `WM_BUTTON1UP` event to the owner of the button.

WM_BUTTON1DOWN. When `button 1` of the mouse is pressed, the mouse is placed in capture mode to allow the monitoring of mouse movement. When `button 1` is released, it is recorded by the button control. The capture is necessary so that if the user moves the mouse pointer outside the limits of the control, all mouse-related events are still received by the button control and the proper notification is performed.

WM_CHAR. Keyboard presses are handled by the button only when

no mouse capture is being performed. The idea here is that actions should not be confused with the function of the control. This situation is similar to serialization of a resource through semaphores. Selection keys such as the spacebar, as well as the Tab and cursor movement keys, are processed.

WM_QUERYDLGCODE. When Presentation Manager is initializing the control, it sends this message to interrogate the control about its capabilities. The button control returns `DLGC_BUTTON` or'ed with one of the values shown in Table 6. These codes allow the Presentation Manager to determine how to handle certain events, including mnemonic selection and decoding.

WM_QUERYWINDOWPARAMS. This message is received by the button control when the `WinQueryWindowText` or `WinQueryDlgItemText` functions are used. The `WPM_TEXT` code is used within the `WNDPARAMS` structure, shown in Figure 12, through the `fsStatus` field. It is also received by the button control when the `WinQueryWindowTextLength` or `WinQueryDlgItemTextLength` functions are used. The `WPM_CCHTEXT` code is used within the `fsStatus` field.

WM_SETWINDOWPARAMS. This message is received by the button control when the `WinSetWindowText` or `WinSetDlgItemText` functions are used. The `WPM_TEXT` code is used within the `WNDPARAMS` structure, shown in Figure 12, through the `fsStatus` field.

WM_PRESPARAMCHANGED. When any of the presentation parameters are changed either by Presentation Manager through a scheme palette change or by the owning application through a `WinSetPresParams` or `WinRemovePresParams` function, the button control forces a repaint of the entire control so the control

reflects the changes made.

WM_ENABLE. The enabling and disabling of the control through the `WinEnableWindow` function causes the button to set an internal flag to allow the control to determine when to ignore keystrokes. It also causes the button to repaint itself to reflect the state change.

WM_SYSCOLORCHANGED. When a user changes the scheme palette for the entire system, this message is received by the button control. The button then causes a repaint to



Use CTLDATA structures that will allow the formal setup of data through the creation of the control as much as possible.

occur, reflecting any changes made to the system colors that may have been used by the button.

WM_MATCHMNEMONIC. If a mnemonic has been provided in the text of the control (designated by a `~` before the character that is to act as the mnemonic), the value passed through the `mp1` message parameter is checked as the designated character. When a match occurs, a value of `TRUE` is returned. However, when no match occurs a value of `FALSE` is returned. Presentation Manager then uses the return value to determine if it should send the `BM_CLICK` message to the button control to simulate a button click.

WM_QUERYCONVERTPOS. The button returns a `QCP_NOCONVERT` to indicate that no conversion should



occur with double-byte character set (DBCS) processing.

GENERAL RULES

The following section summarizes most of the rules you need to be concerned with when creating custom controls. Further information on each topic can be found, where indicated with "(Winter '93)," in the Winter 1993 OS/2 *Developer* article "Demystifying Custom Controls," by Mark Bengé and Matt Smith, and where indicated with "(Spring '93)," in this article.

Use `CTLDATA` structures that will allow the formal setup of data through the creation of the control as much as possible. The information contained within the structure

Something that is three pixels on a VGA system may appear as a dot on a 1280x1024 system.

Maintain a clean and abstract interface so controls that allow the setting of units are based on a range of values. This is similar to the way scroll bars allow you to set the starting and ending points of the range (unlike sliders, which are based on an absolute zero-based system). While this may make the internals of the control more complex, the external use of the control will be easier because the control calculates and reports the position—not the window or dialogue that owns the control.

Maintain a cache of information that allows for speedy painting and keyboard and mouse responsiveness. (Winter and Spring '93)

Keep the control simple and straightforward. Try not to overload your initial controls with functionality by trying to make a control that exceeds all others. Iteratively build up your controls in complexity as you gain experience; it won't take long and they will all work and be useful. (Spring '93)

Keep the control as tight and compact as possible to ensure that it is responsive to user actions. (Spring '93)

When painting, set presentation parameters in RGB mode instead of color index mode. Although it's more difficult, using RGB mode gives you more color flexibility. (Winter '93)

If the control is providing for multiple items, as with a list box, allow for a data pointer that the control owner can reference through a message (for example, list box messages `LM_SETITEMHANDLE` and `LM_QUERYITEMHANDLE`). This makes it easy for users to reference data associated with a selected item.

Presentation parameters are handled on your behalf by Presentation Manager; you need to monitor changing of the parameters only

for use in painting routines.

Fonts and foreground and background colors are handled on your behalf by the owning application window or dialogue. Only areas outside the defaults, such as line type and pattern type, need to be handled. (Winter '93)

When registering a class, reserve room for a private data pointer along with a four-byte pointer area that can be used by the owner of the control. This area should be referenced with the `QWL_USER` manifest constant. The private data for the control should be referenced with `QWL_USER + 4`. The total reserved memory size is eight bytes. (Winter '93)

Ensure that the control has a default style and a default set of limits where appropriate. Assume that the control could be created without any specified styles and limits. An example of this is the Presentation Manager entry field, with its default style of `ES_LEFT` and default text limit of 32 characters.

CONCLUSION

The complexity of a control is constantly misunderstood. As demonstrated with the sample controls in Figure 1, controls can be very simple in purpose and implementation. Don't be intimidated from building a control simply because you think it is complex. A control is just another window, and an application can be thought of as one big composite control.

In a future article, we will develop a control that from which you can select RGB colors, based on the principles of the color wheel. This control will introduce Gpi functions and techniques for clipping, bitmap drawing, and threads to maintain responsiveness. While we haven't yet documented how to use threads in this way, the information will be covered later.

The complexity of a control is constantly misunderstood... controls can be very simple in purpose and implementation.

should correspond to messages that can be sent to the control. This may be important in allowing compatibilities to third-party products and other tools. (Winter and Spring '93)

The `CTLDATA` structure's first word *must* contain the size of the entire structure. This is a cardinal rule that must not be broken within OS/2 2.0. (Winter and Spring '93)

Use dialogue units as the basis of measurement for the control externally and pixels internally. This is important if you want the control to work and look similar over different screen resolutions.

Mark A. Benge, IBM Programming Systems Laboratory, 11000 Regency Pkwy., Cary, N.C. 27512. Benge is a senior associate programmer who joined IBM in 1989. He has worked on record-level I/O for OS/2 and CUA Controls Library/2. He now works in IBM class library user interface development, where he is implementing C++ classes for direct manipulation. Benge received a B.S. in computer science from Western Carolina University.

Matt Smith, Prominare Inc., 100 Front Street East, Toronto, Ontario, Canada, M5A 1E1. Smith is lead architect for the Prominare Development System, an OS/2 2.0 advanced GUI development environment. He has been actively involved with OS/2 since 1988, co-founding Prominare in 1990. Smith has a degree in architecture from the University of Waterloo.

REFERENCES

OS/2 2.0 Programming Guide Volume II (IBM Doc. S10G-6494)

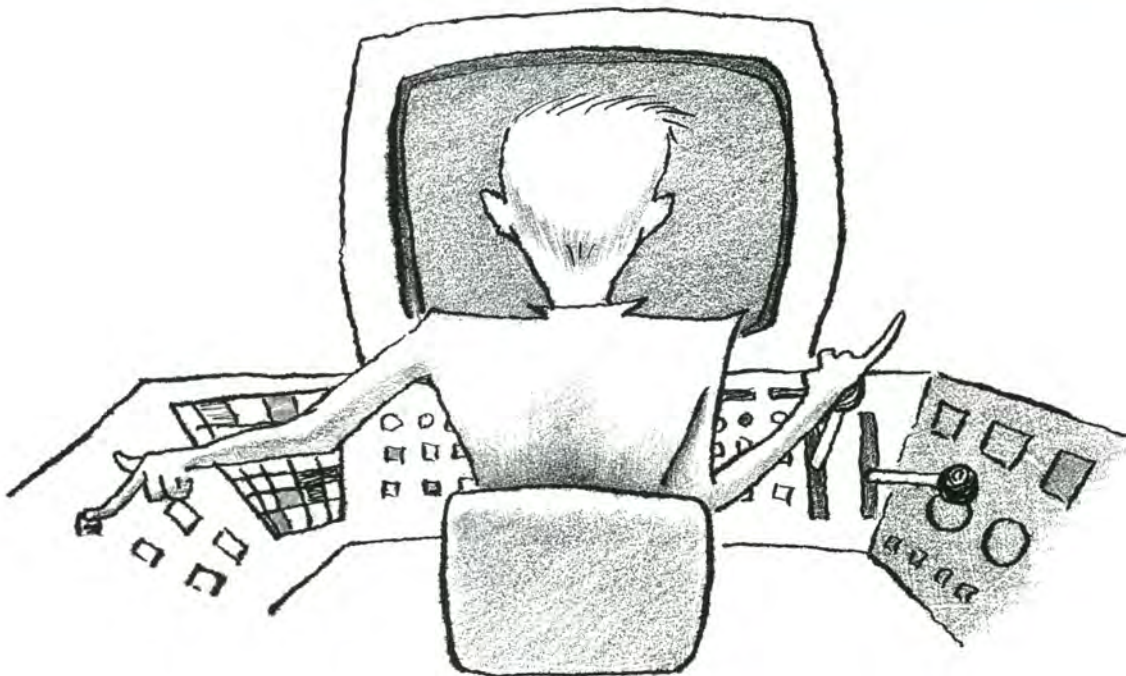
OS/2 2.0 Presentation Manager Programming Reference Volume I (IBM Doc. S10G-6264)

OS/2 2.0 Presentation Manager Programming Reference Volume II (IBM Doc. S10G-6265)

OS/2 2.0 Presentation Manager Programming Reference Volume III (IBM Doc. S10G-6272)

The source code can be downloaded from several electronic sources:

- In the United States, call the IBM PCC BBS at (404) 835-6600. The source code for this article is in File Area 11 under the name CTRLDES.ZIP.
- In Europe, you can find the source code on the International OS/2 Users Group BBS at +44 (0)454 633197 or +44 (0)454 633420.
- On CompuServe, the source code is in LIB13 of the OS2DF2 forum.
- If you have a VM account on an IBM node, you can receive the source code with the command `REQUEST CTRLDES FROM BANZAI AT CARVM3`.





Object-Oriented Programming

This article provides a short, top-level overview of SOM, the IBM System Object Model. Fundamental concepts appropriate to the semantics of SOM and possible SOM extensions are introduced and explained.

BY SCOTT DANFORTH

A Bird's Eye View of SOM



Scott Danforth

DOCUMENTS DESCRIBING THE USE OF SOM on OS/2 2.0 provide complete, detailed coverage of the current SOM product.¹ Various extensions to SOM described here are not supported by an IBM product, but represent an area of current investment in extending SOM technology. These extensions are distinguished from the currently available SOM product with the acronym ESOM.

SOM CAPABILITIES AND OBJECTIVES

SOM supports the definition, construction, and use of object-oriented systems. Although object-oriented programming languages provide these capabilities, SOM does so in a way that is independent of programming languages by defining an application programming interface (API) to SOM objects based on simple procedure calls. SOM's objective is not to replace existing programming languages, but, rather, to allow applications written in different programming languages to use a common class library and allow such libraries to be extended without affecting existing applications.

A SOM class library is distributed in binary form as a dynamic link library (DLL) that can be replaced without requiring recompilation of applications (as long as the new library doesn't require changes in the applications' source code). This is the usual situation with procedure-based libraries, but SOM extends the capability to object-oriented class libraries. This is essential if system-provided,

object-oriented application frameworks are to be used by software vendors, since users will not have access to application source code for recompilation when new releases of a system library are installed. Language neutrality and upward binary compatibility are two reasons why SOM was chosen for building the OS/2 Workplace Shell.²

SOM need not be used to implement all of the objects and classes important to an application. Many objects created and used by an application will be strictly internal to that application's implementation—with no need for access by multiple languages or applications. Perhaps these internal objects will be SOM objects (in those cases where a SOM class library provides useful functionality), or perhaps they will be objects provided by some other class library (for example, a Smalltalk or C++ class library), depending on the language used to program the application. Only objects intended to be upwardly binary-compatible across application and language boundaries (or useful to non-object-oriented languages) need to be SOM objects. Supporting such objects is SOM's main purpose.

The SOM run time consists primarily of four objects that are implemented using SOM. Thus, SOM itself derives benefits offered by binary class libraries, and new releases of a SOM dynamic link library (DLL) can provide increased functionality while still supporting existing application binaries. For



example, ESOM supports the Object Management Group CORBA (common object request broker architecture) standard³ and includes a variety of features not available from the original OS/2 2.0 SOM used by the Workplace Shell. (Differences include multiple inheritance classes and support for CORBA-standard interoperability between applications on different machines in distributed environments.) Yet, the single inheritance SOM DLL can be replaced with an ESOM DLL in OS/2 2.0 without affecting the Workplace Shell or any other applications that depend on SOM.

SOM TERMINOLOGY AND APPROACH

Objects. A SOM object is a run-time entity with a specific set of associated methods and instance variables. An object's implementation determines what these methods are (specifying procedures to implement the desired method semantics), what the instance variables are, and the overall organization and placement of method procedures and instance variables within the object. The procedures and instance variables used to implement an object's semantics are encapsulated by the object and are not directly accessible to an object's user. Instead, an object's user is given access to selected methods and instance variables through an interface.

Types. Unlike many current approaches to object-oriented programming, SOM distinguishes types (which correspond to interfaces) from classes (which correspond to implementations). Although the value of separating these concepts is generally understood by researchers in object-oriented programming,⁴ readers familiar with current object-oriented programming languages may be used to equating these concepts. Therefore, we will take special care in introducing an appropriate and consistent terminology.

An object type represents a specific interface through which an object's methods or instance variables may be accessed. By declaring a type for an object (for example, a procedure argument), a programmer chooses

the interface that will be used when accessing that object.

In object-oriented programming, a variety of different interfaces may be appropriate for a given object. This is why objects in object-oriented systems are characterized as being polymorphic—that is, of many forms. Type-checking is the act of verifying that the interface chosen by a programmer for accessing an object is actually appropriate.

Classes. In SOM, as in most approaches to object-oriented programming, classes define the implementation of objects, and every SOM object is an instance of a single, specific

A SOM object is a run-time entity with a specific set of associated methods and instance variables.

SOM class. All SOM classes are derived by subclassing—either directly or indirectly—from the primitive root class called **SOMObject**, which introduces useful methods common to all SOM objects.

A class in SOM is an object (called a class object) which may correspond to a static specification. When it exists, this specification, called a class declaration, specifies how the class is derived—by indicating parent classes (whose instance's methods, supporting procedures, and instance variables are inherited) and by indicating new methods and instance variables introduced by the class.

Because classes in SOM are objects, subclassing is a dynamic, run-time activity. Although a class declaration indicates parent classes and the methods that will be supported by the class's instances, it is the run-time class object that determines (during an initialization process that includes inheritance from parent class objects) the actual procedures that will implement methods on its instances.

After it has been initialized, a class object is ready to create class instances, that is, objects.



Once an object is created, an interface must be used to invoke its methods or access its instance variables. As mentioned previously, object types represent these interfaces.

THE SOM COMPILER

Types are determined by the SOM compiler, which processes class declarations expressed in object interface definition language (OIDL). When a SOM class is declared, each introduced aspect (that is, the method or instance variable) may be public, private, or

introduced types into derived types may be viewed as type inheritance or subtyping (in contrast with implementation inheritance or subclassing). In SOM, there are two derived types corresponding to each derived class: the public derived type and the private derived type (or, simply, the public type and the private type).

The public type corresponding to a class represents an interface that allows access to all the public methods and instance variables introduced by the class as well as those introduced by its ancestor classes. The private type corresponding to a class represents an interface to all aspects made visible by the public type, plus the private methods and instance variables introduced by the class and, possibly (based on information expressed in the class declaration), the private methods and instance variables included in selected parents' private types.

There are a number of reasons why SOM supports a distinction between public and private types. Among them is the fact that public methods and instance variables may never be retracted. Object clients who rely solely on public types will never require source changes (and, thus, will never require recompilation). If SOM were used to create an application framework, for instance, application code should use only the public framework types, but different framework classes that are tightly coupled with respect to implementation might want to use private types.

To allow SOM objects to be used by different programming languages, the SOM compiler maps the public and private types for a given SOM class to language-specific bindings that provide the corresponding interfaces to objects of that class. Although language bindings (and the interfaces to SOM

objects that they embody) take different forms depending on the language involved, these bindings all rely on use of run-time data structures that are initialized when class objects are created. These data structures are an essential component of the SOM API and correspond to the introduced types for classes.

SOM specifies a particular mapping of class declarations to introduced types and a particular mapping of introduced types to a corresponding SOM API based on run-time data structures. The SOM compiler then generates language-specific bindings that use the SOM API to implement the interfaces represented by derived types in a way appropriate to the language being supported.

Programmers need not be aware of the SOM API, since language bindings provide an API to SOM objects appropriate to the specific language involved. The original SOM product provides C bindings. In ESOM, C++ bindings for SOM classes allow a C++ programmer to use SOM objects as if they were C++ objects, and the C++ compiler provides complete typechecking of SOM object use. With Smalltalk bindings in ESOM, SOM objects appear as Smalltalk objects, and the Smalltalk browser can be used for browsing and dynamic inspection of SOM classes.

The SOM Run Time. The SOM implementation has static and dynamic aspects. The static aspect is represented by the SOM compiler, which supports the SOM object model by parsing OIDL class declarations and producing language-specific bindings for SOM classes. The dynamic aspect is represented by a run-time environment that consists of useful procedure entry points and four SOM objects. The SOM run time is used by the language bindings and is also available to

Types are determined by the SOM compiler, which processes class declarations expressed in object interface definition language (OIDL)

internal. Based on this information, two different introduced types are recognized by the SOM compiler. These types represent the public and private interfaces to the aspects of an object that are introduced by its class (internal aspects of an object are hidden, and no interface to them is provided).

Interfaces for methods and instance data inherited from a class's ancestors are not represented by introduced types. To support the polymorphism inherent to object-oriented programming, the introduced types for a class are combined with ancestor classes' introduced types to create derived types that represent a union of interfaces appropriate to class instances. This process of combin-

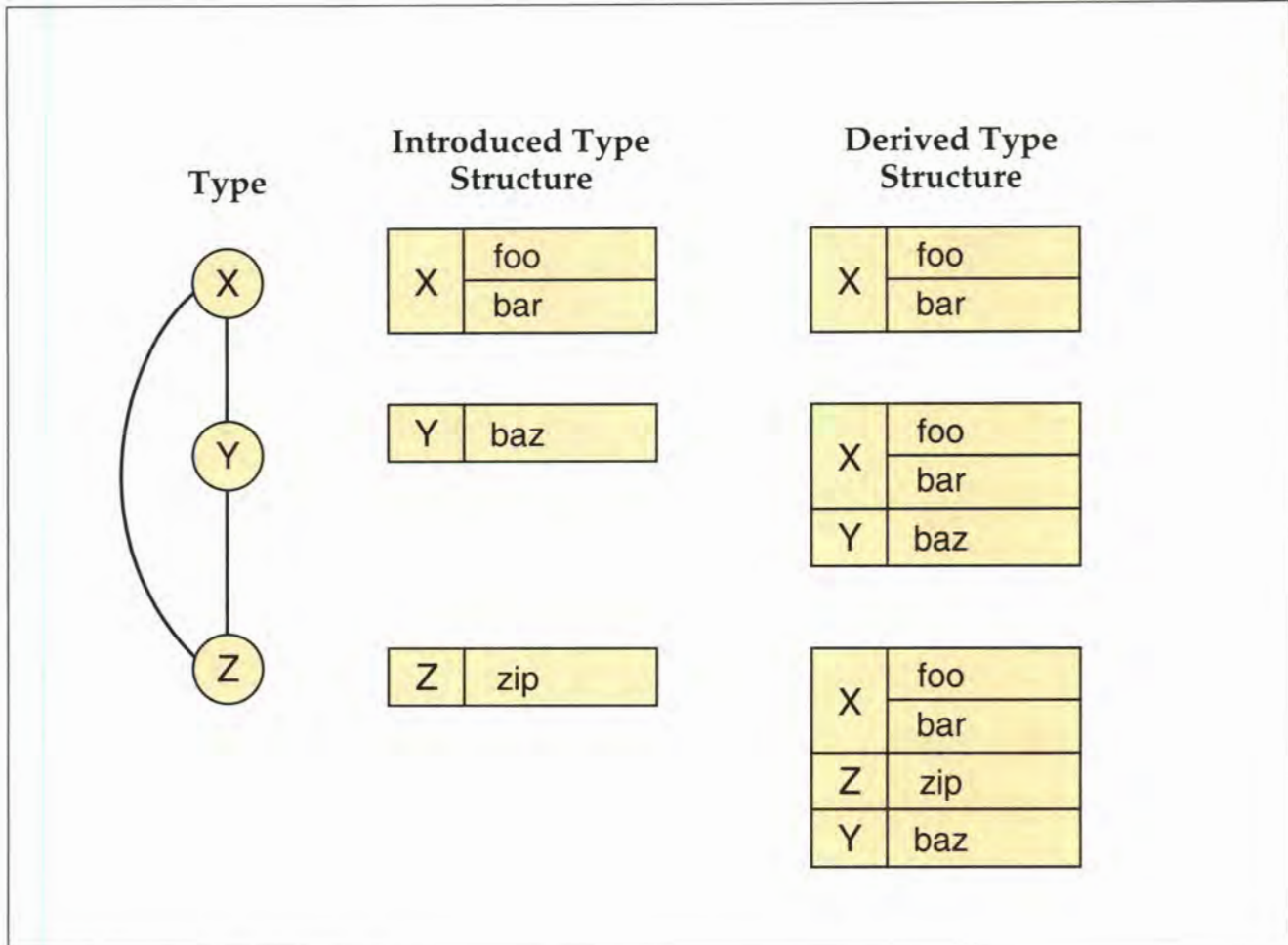


Figure 1. Introduced and derived object types

client programs.

Of the four SOM objects provided by the SOM run time, three are class objects and one is a class manager whose purpose is to dynamically load and register new SOM classes into a system. The class objects are *SOMObject* (the class of all SOM objects), *SOMClass* (a metaclass—that is, a class whose instances are SOM classes), and *SOMClassMgr* (the class of the class manager).

The three primitive classes provided by the run time and the class manager object are central to the operational semantics of SOM. They provide the main portions of the SOM API.

INHERITANCE IN SOM

One of the defining aspects of an

object model is its characterization of inheritance. In SOM, inheritance relates to types and classes. Support for inheritance of interface (that is, subtyping) is mainly the

concern of metaclass objects (for example, *SOMClass*) that define the methods that create and initialize class objects. These methods must guarantee that an interface appropriate

One of the defining aspects of an object model is its characterization of inheritance.

concern of the language bindings produced by the SOM compiler, which must guarantee that an interface available on instances of a parent class will also be available on instances of its subclasses. Support for inheritance of implementation (subclassing) is mainly the con-

cern of metaclass objects (for example, *SOMClass*) that define the methods that create and initialize class objects. These methods must guarantee that an interface appropriate

Following are a few examples to help clarify the semantics of inheritance in SOM. For simplicity, the distinction between public and private types is not reflected in these



examples, and interfaces to instance variables are not explicitly considered. These details are not essential to a general understanding of SOM, and are documented elsewhere.

Type Inheritance—Subtyping. A derived type is used to represent an overall interface to objects of a given class and has a structure that can be characterized as a discriminated union of introduced type structures inherited from ancestor types.

In Figure 1, two SOM object types and a multiple-inheritance ESOM object type are introduced: respectively, X, Y, and Z. The type X

This model seems the most natural approach to multiple inheritance.

This example uses only methods, but, in general, selected instance variables may also be included in an introduced type. As with methods, interfaces to instance variables are also grouped in a derived type according to their introducing type and appear only once (as required by graph inheritance).

The SOM API for method resolution requires as arguments an object and method token. The method token specifies an introducing type and an offset relative to the beginning of that type's introduced method group. This

variable groups that correspond to type interfaces. It is primarily the language bindings that support types, which implies that there must be some agreed-upon protocol between code that creates and initializes a class object (and, in the process, receives method and data tokens) and code that wants to use instances of this class. SOM defines this protocol, specifying the runtime data structures in which method and data tokens are stored. Language bindings support and rely on this protocol.

Implementation Inheritance—Subclassing. SOM classes define the derivation structure reflected by type inheritance, and this structure is ultimately realized in the organization of class instances and instance method tables. This organization comes about through inheritance of implementation—specifically, the implementation of class instances.

Implementation inheritance takes place in SOM during subclassing, when an initialization method is invoked on a newly created, uninitialized class object. (Uninitialized class objects are created by invoking the *new* method on a metaclass—that is, a class object whose instances are class objects.) Three essential pieces of information are provided to the initialization method for a new class: a list of parent classes for the new class (whose methods are then inherited), the number of additional methods to be introduced by the new class, and the amount of space required by the instance variables introduced by the new class. After initializing the class, the class implementor uses additional method calls to override inherited methods, add procedures that support newly introduced methods, and load API data structures with method and data tokens for newly introduced methods and data.

SOM classes define the derivation structure reflected by type inheritance, and this structure is ultimately realized in the organization of class instances and instance method tables.

introduces two methods, called *foo* and *bar*. The type Y is derived from X's type by single inheritance and introduces a new method, called *baz*. The derived type structure for Y is the union of the introduced types for X and Y, and methods are explicitly grouped according to their introducing type. The type Z is derived by inheriting from X and Y and by adding the method called *zip*. The organization of the introduced types within Z's derived type reflects ESOM's implementation of multiple inheritance.

Although X appears twice in Z's derivation (once directly and once indirectly through Y), X's introduced method group only appears once in Z's derived type. This is consistent with the CORBA model for multiple inheritance and has been called "graph inheritance."⁵

information is used by SOM to access the instance method table and return a pointer to the procedure that supports the indicated method. The API for instance data access is similar, requiring as arguments an object and data token. Method and data tokens are part of the interface to objects that are represented by introduced types.

Where do method and data tokens come from? When a new method is added to a class's instance method table during the initialization of class objects, the returned result is a method token for the new method. In the case of instance variables, also, class objects provide the necessary support.

SOM class objects know nothing about public or private interfaces and instance data other than how to find (within objects) the instance

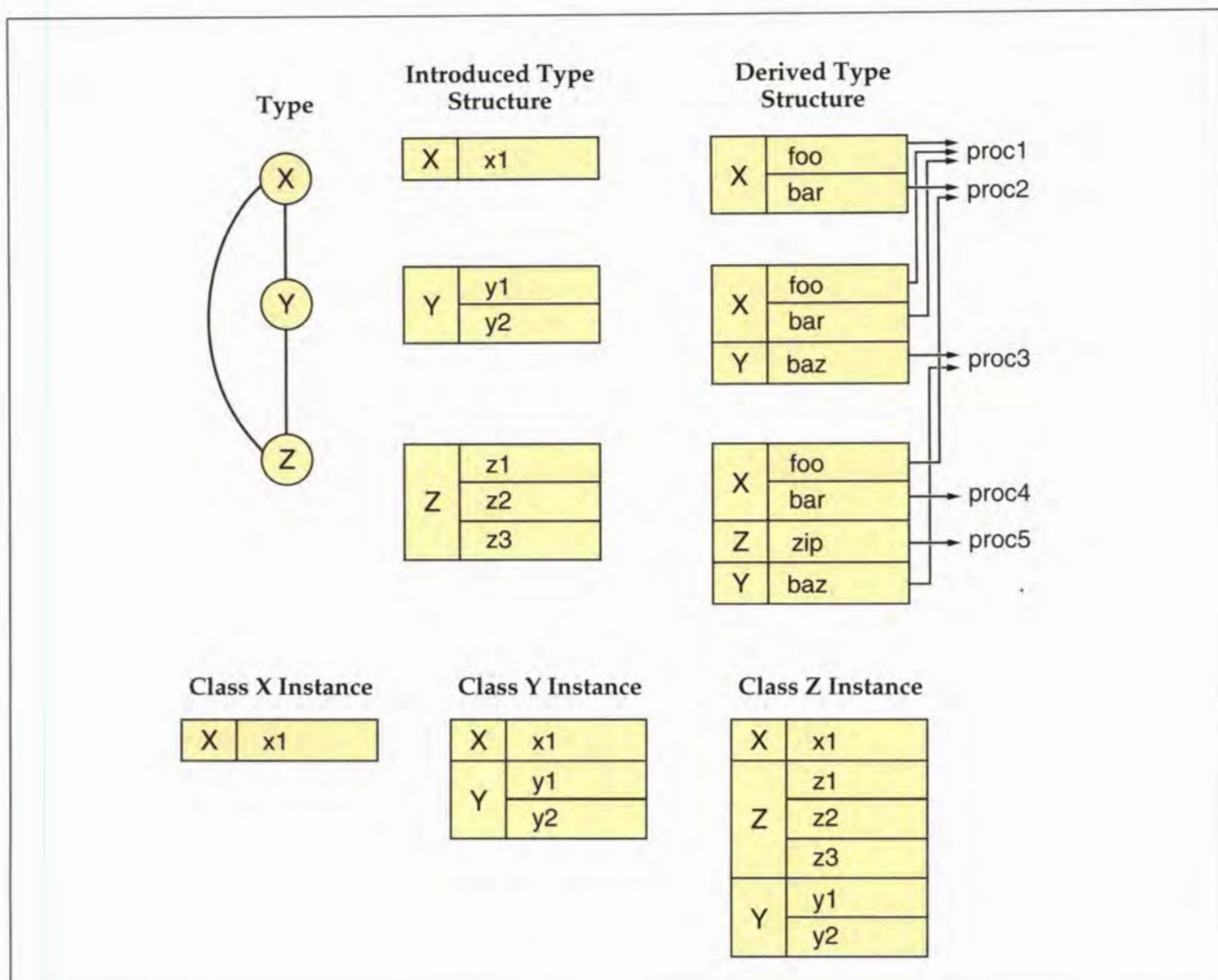


Figure 2. SOM and ESOM class derivations

Subclassing initializes two instance variables of a class object: the instance method table and the instance size. The instance method table is a table of pointers to the procedures that implement the methods available on the class's instances. The instance size indicates the amount of storage required to hold an instance (that is, the space needed to hold an individual object's instance variables).

At this point, an example of subclassing is called for. This is provided by the SOM and ESOM class derivations in Figure 2, which correspond to the type derivations in Figure 1. The essential information

provided by an implementation concerns the procedures that support methods on a given class of objects and the instance variables that support these procedures. Figure 2 shows the derived classes' instance method tables, which contain pointers to the procedures that implement methods for these classes and shows how the instance variables introduced by each derived class appear in the classes' instances.

The example in Figure 2 assumes that Y overrides the method bar (inherited from X) with the procedure *proc3*. The instance method tables for each class correspond to the derived types for

these classes, as shown in Figure 1. Thus, given an appropriate type for an object, it is possible to correctly invoke methods on the object. Also, instance variables are grouped in the same way as methods, and that only one copy of X's instance variables is found in Z's objects.

Figure 2 illustrates how ESOM resolves a particular form of ambiguity that arises when using multiple inheritance. When a method is inherited from multiple parents of a derived class, the procedure used to support that method is the one used by the leftmost parent from which the method is inherited. Thus the procedure *proc2* is used when the method bar is invoked on

an instance of Z. Of course, the default can be overridden if this is desired (in this case, for example, Z could decide to use `proc4` or some other procedure to the support bar).

As illustrated by this example, the absence of published (that is, public or private) instance variables in the introduced types for X, Y, and Z does not mean that the implementation of the corresponding classes requires no instance variables. It just means that the instance variables illustrated in Figure 2 are internal and will only be accessible to the procedures that implement classes' methods. The X, Y, and Z introduced types in Figure 1 are called pure types. A pure type is an introduced type that publishes no instance variables and has only pure ancestor types.

Abstract Inheritance. In ESOM, different classes can implement objects that support the same type even though they embody entirely different implementations. This is achieved by providing a class initialization method that inherits only the instance method table structure necessary to support a parent's derived types—method procedures and instance variables are not inherited. Abstract inheritance is available only from classes whose introduced types are pure. A published interface to a given instance variable requires that the instance variable be inherited.

The value of abstract inheritance is that it allows complete replacement of implementations without affecting subclasses or client code and without the need to define additional, auxiliary abstract classes. Because instance data is not inherited, a significant reduction in the size of objects may be achieved. Of course, when abstract inheritance is used, the implementor of the new class must provide new instance variables and method pro-

cedures for implementing inherited methods. But this is the point of abstract inheritance; the intention is to allow the implementation of a subclass to be different from that of its parent and to be based on the use of different instance variables. If inheritance of instance variables is desired, then implementation inheritance (that is, normal subclassing) should be used.

To allow selected portions of a class of objects' instance variables to be inherited, abstract and implementation inheritance can be combined, as in the example in Figure

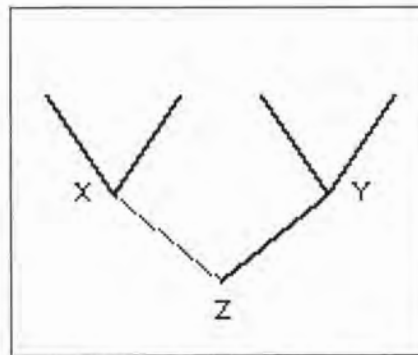


Figure 3. Combining abstract and implementation inheritance

3. Abstract inheritance is shown using a dotted line, and solid lines are used to indicate implementation inheritance. In this example, the class Z chooses to replace the implementation for X objects that would otherwise be inherited. The result is that instances of Z (or of classes derived from Z) will all support X's derived types even though these objects are not based on X's implementation and include none of X's (or X's ancestors') instance variables.

The assumptions of this example are that X's introduced types are pure, and that Z introduces its own support for the methods inherited from X. In contrast, Z inherits implementation from Y. Implementation inheritance takes precedence over abstract inheritance, so Z will inherit its implementation from any class

that is an ancestor of X and Y.

The ability to combine implementation inheritance with abstract inheritance allows flexible and precise control over the implementation of objects. For example, if the class Z in Figure 2 were to use abstract inheritance from Y and implementation inheritance from X, then Z would inherit all of X's implementation and none of Y's. The changes required to the diagram would be the use of a dotted line from Y to Z (to indicate abstract inheritance), Z would override `baz` (to provide an implementation for this method), and the illustrated Z instance would include no instance variable group for Y. Of course, a Z instance would still support all of Y's interfaces.

CONCLUSION

Some advanced SOM capabilities were not discussed in this paper, including the use of derived metaclasses to provide inheritance semantics tailored to special uses, and the use of SOM's dynamic dispatch interfaces to provide enhanced method invocation semantics. Nevertheless, this article has included the essential aspects of the SOM approach and shown how SOM provides an advanced foundation for object-oriented systems development.

SOM does not try to support all the capabilities of all object-oriented programming languages, but SOM does provide a concrete foundation upon which advanced object-oriented systems and language-neutral class libraries can be based. In addition, a primary focus of SOM technology concerns the ability to change or even replace object implementations without affecting existing applications. As mentioned previously, this ability has been useful to SOM, since much of the SOM run time is implemented as SOM objects.

The main contribution of SOM is

its support for language-neutral, upward-compatible binary class libraries. In contrast, language-centric class libraries are limited in utility (they are useful only from a single language) and require publication of source code. Different compilers for the same object-oriented language cannot be guaranteed to use the same object layouts in memory, so binary class libraries produced using one compiler will generally be useless to applications developed with a different compiler.

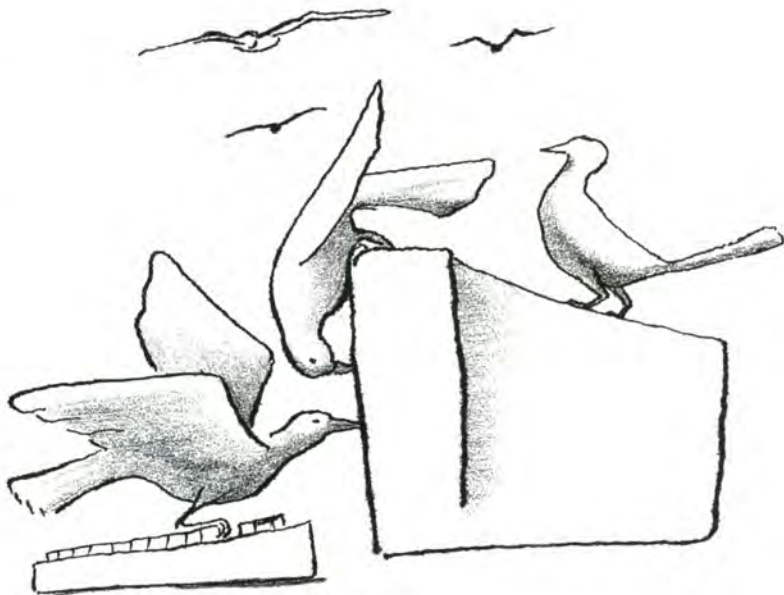
ACKNOWLEDGMENTS

Mike Conner was the originator of SOM. Realization of the SOM and ESOM run times is due to the efforts of Mike Conner, Andy Martin, Larry Raper, and Nurcan Coskun. ESOM Smalltalk bindings are due to Ken Murray, and assistance with abstract inheritance was provided by Roger Sessions. The SOM project as a whole has benefited from the guidance of IBM Fellow Larry Loucks.

Scott Danforth, IBM Corp., 11400 Burnet Rd., Austin, Texas 78758, (512) 838-8074. Danforth received his Ph.D. in computer science from the University of North Carolina at Chapel Hill, where he assisted in the design of a cellular architecture for parallel execution of functional languages. Currently, he is developing language-neutral object technology for binary class libraries, and system-level frameworks for object-oriented programming. He is on the editorial board of the *International Journal of Parallel Programming*. His interest areas include parallel processing, advanced programming languages, and databases.

REFERENCES

1. Bucko, M., and L. Raper. *OS/2 2.0 Technical Library System Object Model Guide and Reference*, 1992. (IBM Doc. 10G6309)
2. *OS/2 Version 2.0 Volume 4: Application Development*, 1992. (IBM Doc. GG24-3774-00)
3. *The Common Object Request Broker: Architecture and Specification*. The Object Management Group Inc. (DEC Corp., Hewlett-Packard Co., HyperDesk Corp. NCR Corp., Object Design Inc., and SunSoft Inc.) and x/Open), 1991.
4. Harris, W. "Contravariance for the Rest of Us," *Journal of Object-Oriented Programming*, 4(7): Nov. 1991.
5. Snyder, A. "Encapsulation and Inheritance in OOP Languages," *OOPSLA Conference Proceedings*, 1986.





Object-Oriented Programming

Object-oriented programming is completely changing the way we write software. SOM (System Object Model), IBM's advanced object-oriented programming and packaging technology, is at the forefront of this change.

BY ROGER SESSIONS and NURCAN COSKUN

Method Resolution in SOM

POLYMORPHISM, THE ABILITY TO RESOLVE methods, is the basic feature of all object-oriented programming systems. It allows a given method name to be defined in several places. Object-oriented systems use method resolution to map method invocations to actual procedures. SOM is an object-oriented system that is unusual in that it allows class designers and programmers considerable latitude in deciding how methods are resolved.



Roger Sessions

INTRODUCTION

Object-oriented programming employs three major features: encapsulation, which allows you to write multi-purpose and easily tested classes; inheritance, which improves your ability to reuse code; and polymorphism, which allows code to be simplified and maintenance costs to be reduced. For a good introduction to object-oriented programming, see Roger Sessions' *Class Construction in C and C++* (Prentice-Hall, 1992).



Nurcan Coskun

SOM is an important new technology designed to significantly enhance existing object-oriented programming languages, and bring object-oriented capability to those languages that are essentially procedural. The most important feature of SOM and the one that most sets it apart from other object-oriented technologies is its language neutrality. SOM is designed to allow classes written in one programming language to be used with any other language. When we say that these classes can be used, we mean it in the fullest

possible object-oriented sense: the classes can be instantiated, manipulated, and, most importantly, subclassed from languages other than the language in which they were written. The only requirement is that both languages support SOM bindings.

The first of these bindings, the C bindings, are already available in the OS/2 2.0 Toolkit. Additional language bindings under development include C++, Smalltalk, and COBOL. IBM is encouraging many different language vendors to directly support SOM bindings while working with several standardization bodies to make SOM as widely accepted as possible.

One of the features of SOM that allows bindings to be created for so many different languages is its concept of flexible method resolution. This same flexibility can be exploited by the SOM programmer to choose the resolution most appropriate for a given task. Before we discuss the techniques used in method resolution, let's briefly review the basic idea of method resolution. The discussion here assumes familiarity with object-oriented programming concepts and SOM. For an introduction to either of these areas, see the previously mentioned references.

Consider this SOM class definition for a dog:

```
class: dog;  
parent: SOMObject;  
methods:  
    void bark();
```




Assume that bark() for dog is implemented as: WOOF WOOF

```
static void bark(dog *somSelf)
{
    dogMethodDebug("dog", "bark");
    printf("woof woof");
}
```

Consider another SOM class bigDog defined as:

```
class: bigDog;
parent: dog;
methods:
    override bark;
    and reimplementing bark() as:
static void bark(bigDog *somSelf)
{
    bigDogMethodDebug("bigDog", "bark");
    printf("WOOF WOOF");
}
```

Now, suppose we have client code, such as:

```
...
dog *myDog;
if (dogToGet() == DOG) myDog = dogNew();
else myDog = bigDogNew();
_bark(myDog);
...
```

In this code, myDog may end up being a dog, or it may end up being a bigDog; there is no way to tell at compile time. Since we don't know which class myDog will be, we also don't know how the line:

```
_bark(myDog);
```

should be interpreted. Should it assume myDog is a dog, since that is how the pointer was originally declared? If so, then the method should invoke dog bark() and print:

```
woof woof
```

But this might well be the wrong method. At run time, myDog may be pointing at an object of type bigDog. If myDog is really pointing to an object of type bigDog, then the correct bark() method is the one associated with bigDog, and the correct output is:

Although this example is trivial, it points out that method resolution is not as simple as it first appears. We might naively assume that client code, such as:

```
_bark(myDog);
```

will simply map to an existing function with the same or similar name. As shown here, the problem is much more complex. There may

SOM's concept of flexible method resolution allows bindings to be created for many different languages.

be many candidate functions, and it is the job of method resolution to determine at run time which should be invoked.

SAMPLE SOM CLASS

Before we discuss method resolution, let's look at a typical class that will be used throughout the remainder of this article. The SOM definition of a student class is shown in Figure 1. A student has an id and a name and supports the setUpStudent() method, among others. The implementation of the student class is shown in Figure 2.

Given this class, the client code:

```
Student *student1 = StudentNew();
_setupStudent(student1, "599600",
"David Brown");
_printStudentInfo(student1);
```

would print the output:

```
Id: 599600
Name: David Brown
Type: student
```

The invocation of the method setUpStudent()

is obviously resolving to the function shown in Figure 3.

In the rest of this article, we will look more closely at how this resolution occurs and the various resolution options available to the C programmer.

OFFSET RESOLUTION

We can learn a lot about how SOM operates by saving the C preprocessor-generated output from the client program. For details on how to accomplish this, see the reference material for your particular C compiler. To simplify our examination, we will also turn off SOM run-time type checking. For details on this, refer to the *Technical Library Guide* for SOM, listed in the References section.

We need to look at preprocessor generated output to see the effect of expanding macros; what we think of as the method `setUpStudent()` is actually a macro defined in `student.h`. The client code line:

```
_setUpStudent(student1, "599600",  
"David Brown");
```

expands into the code shown in Figure 4. This expanded code accomplishes the following steps:

1. It calls the function `somResolve()` with the parameters `student1` and `StudentClassData.setUpStudent`. The first parameter, `student1`, is a pointer to the target object. The second parameter, `StudentClassData.setUpStudent`, is a member of a structure that essentially gives the offset within a method table for the method `setUpStudent()`. The method table, a table of pointers to functions, very similar to the virtual mechanism used by C++ (described in *Class Construction in C and C++*).

2. `somResolve()` returns a pointer to the appropriate function to map onto this method invocation. In this

```
include <somobj.sc>  
  
class:  
    Student;  
  
parent:  
    SOMObject;  
  
data:  
    char id[16];      /* student id */  
    char name[32];    /* student name */  
  
methods:  
    override somInit;  
  
    void setUpStudent(char *id, char *name);  
    -- sets up a new student.  
  
    void printStudentInfo();  
    -- prints the student information.  
  
    char *getStudentType();  
    -- returns the student type.  
  
    char *getStudentId();  
    -- returns the student id.
```

Figure 1. Definition of the student class `student.csc`

```
#define Student_Class_Source  
#define SOM_NoTest  
#include "student.ih"  
  
SOM_Scope void    SOMLINK somInit(Student *somSelf)  
{  
    StudentData *somThis = StudentGetData(somSelf);  
    parent_somInit(somSelf);  
    _id[0] = _name[0] = '\0';  
}  
  
SOM_Scope void    SOMLINK setUpStudent(Student *somSelf,  
    char *id, char *name)  
{  
    StudentData *somThis = StudentGetData(somSelf);  
    strcpy(_id, id);  
    strcpy(_name, name);  
}
```

Figure 2. The implementation of the student class `student.c` (continued on page 97)



```
SOM_Scope void SOMLINK printStudentInfo(Student *somSelf)
{
    StudentData *somThis = StudentGetData(somSelf);
    printf("\n Id      : %s \n", _id);
    printf(" Name     : %s \n", _name);
    printf(" Type      : %s \n", _getStudentType(somSelf));
}

SOM_Scope char * SOMLINK getStudentType(Student *somSelf)
{
    StudentData *somThis = StudentGetData(somSelf);
    static char *type = "student";
    return (type);
}

SOM_Scope char * SOMLINK getStudentId(Student *somSelf)
{
    StudentData *somThis = StudentGetData(somSelf);
    return (_id);
}
```

Figure 2. The implementation of the student class student.c (continued from page 96)

```
SOM_Scope void SOMLINK setUpStudent(Student *somSelf,
                                     char *id, char *name)
{
    StudentData *somThis = StudentGetData(somSelf);
    strcpy(_id, id);
    strcpy(_name, name);
}
```

Figure 3. setUpStudent() for Student class

```
(( ( somTD_Student_setUpStudent ) somResolve((student1),
StudentClassData.setUpStudent )) (student1,"599600","David Brown"));
```

Figure 4. Expanded code for setUpStudent() invocation

case, the returned pointer points to setUpStudent() as implemented in student.c.

3. The returned pointer is typedefed to be a pointer to a function of type somTD_Student_setUpStudent, which is essentially a function that takes three parameters, the first of type student, the second of type char *, and the third of type char *. We

typecast the pointer to ensure that the C compiler will deal properly with the types of the parameters.

4. The function being pointed to is then called and passed the parameters student1, 599600, and David Brown. Because of the typecasting described in the previous step, we can be sure these parameters are of the right number and type. A mistake

here will result in an error at compile time.

We can see why this type of method resolution is called offset resolution: the setUpStudent member of the structure StudentClassData contains an offset within a method table that enables us to look up the desired function very quickly. Of course, this is all happening under the covers; the only thing the client program sees is the line:

```
_setUpStudent(student1, "599600",
"David Brown");
```

Offset resolution is the default for resolving methods in SOM. It is the fastest form of resolution, and, in most cases, gives the desired behavior. In some situations, however, offset resolution is not useful, and other options, such as name-lookup resolution, must be used.

NAME-LOOKUP RESOLUTION

As we have seen, offset resolution is essentially a two-stage process for resolving methods. In stage one, we give an offset to the function somResolve() and are returned a pointer to a function. In stage two, we use that pointer to actually call the function.

Conceptually similar to offset resolution, name-lookup is also a two-stage process. Stage one also involves looking up a pointer to a function, but now instead of looking up by offset, we look up by the name of the method (hence the name "name-lookup"). There are two other important differences. Instead of looking up via the function somResolve(), we use the class method _somFindMethod(), which is defined on SOMClass. (For a detailed description of the SOMClass class, see "Class Objects in SOM," listed in the References section.) In addition, this lookup is not done under the covers, but rather handled directly


```

#include "student.h"

typedef void (*methodType1) (SOMObject *, char *, char *);
typedef char * (*methodType2) (SOMObject *);

/*
-----
Function to invoke method returning nothing
and expecting two string parameters.
----- */
sendMessage1(char *method, SOMObject *target, char *str1, char *str2)
{
    somMethodProc *methodPtr;
    _somFindMethod(_somGetClass(target), somIdFromString(method),
                  &methodPtr);
    ((methodType1)methodPtr)(target, str1, str2);
}
/*
-----
Function to invoke method expecting no
parameters and returning string.
----- */
char *sendMessage2(char *method, SOMObject *target)
{
    somMethodProc *methodPtr;
    _somFindMethod(_somGetClass(target), somIdFromString(method),
                  &methodPtr);
    return(((methodType2)methodPtr)(target));
}
/*
-----
Demonstration program.
----- */
main()
{
    Student *student1 = StudentNew();
    Student *student2 = StudentNew();
    char *id, *type;

    /* Offset resolution
    ----- */
    _setUpStudent(student1, "599600", "David Brown");
    _printStudentInfo(student1);

    /* Name lookup resolution
    ----- */
    sendMessage1("setUpStudent", student2, "120045", "Janet Smith");
    _printStudentInfo(student2);

    id = sendMessage2("getStudentId", student2);
    printf("Student Id = %s \n", id);

    type = sendMessage2("getStudentType", student2);
    printf("Student Type = %s \n", type);
}

```

Figure 5. Demonstration of name lookup resolution


```

#include "student.h"
#include <stdio.h>
#define MAX_LENGTH 80

void getLine(char *response, char *msg); /* Print msg, get response */
main()
{
    /* Local Declarations.
    ----- */
    int i, argNo;
    char *str, *arg, *argList;
    char line[MAX_LENGTH];
    char method[MAX_LENGTH];
    char msg[MAX_LENGTH];
    Student *student = StudentNew();

    /* Display interpreter menu.
    ----- */
    getLine(line, "\n> Menu Options are 'm(message)' , 'q(quit)'\n$");
    while(line[0] == 'm'){

    /* Ask for the method name, number of arguments, and actual argu-
    ments.
    -----
    - */
    getLine(method, "> Enter method name:\n$");
    getLine(line, "> Enter the number of arguments:\n$");
    argNo = atoi(line);
    argList = (char *) malloc((sizeof (char *)) * argNo);
    for(i=0; i < argNo; ++i){
        sprintf(msg, "> Enter argument[%d]:\n$", i);
        arg = (char *) malloc(MAX_LENGTH);
        getLine(arg, msg);
        *(char **)(argList + (i * sizeof(char *))) = arg;
    }

    /* Find out what type the method returns.
    ----- */
    getLine(line, "> Enter the return type:\n$");

    /* Dispatch the method and display result.
    ----- */
    if(line[0] == 's'){
        str = (char *) SOMObject_somDispatchA(student,
            somIdFromString(method), "", argList);
        printf("\n==> Return string from <%s> method is : <%s>.\n",
            method, str);
    }
    else{
        SOMObject_somDispatchV(student, somIdFromString(method), "",
            argList);
    }
}

```

by our code.

Compare this code, using offset resolution:

```

_setupStudent(student1, "599600",
"David Brown");

```

to the equivalent code, using name-lookup resolution:

```

typedef void (*methodType)
(Student *, char *, char *);
somMethodProc *methodPtr;
_somFindMethod(_somGetClass(stu-
dent1), somIdFromString("setUpStu-
dent"), &methodPtr);
((methodType) methodPtr)
(student1, "599600", "David Brown");

```

It is not immediately obvious why anybody would want to use name-lookup resolution when offset resolution is clearly much easier and presumably much faster; there are, however, several reasons. In general, name-lookup resolution is used when you know the the *signature* of the method to be called at compile time but will not know which *method* to call until run time. The code in Figure 5 demonstrates such a program.

NAME-LOOKUP RESOLUTION AT CLASS DEFINITION

Name-lookup resolution can also be made to occur under the covers, as with offset resolution. This is done by adding the modifier *Name Lookup* to the method when the method is prototyped in the .CSC file. For example, if we change the line in the student.csc file from:

```

void setupStudent(char *id,
char *name);

```

to:

```

void setupStudent(char *id,
char *name), name lookup;

```

we find that the preprocessor-generated output of the client code line:

Figure 6. Student class interpreter code (continued on page 100)


```
_setUpStudent(student1, "599600",
"David Brown");
```

changes from:

```
(( ( somTD_Student_setUpStudent )
somResolve((student1),
StudentClassData.setUpStudent ))
(student1, "599600", "David Brown"));
```

to:

```
(( ( somTD_Student_setUpStudent )
(( (somTD_SOMClass_somFindSMethodOk )
somResolve(((student1->mtab)->
classObject)),
SOMClassClassData.
somFindSMethodOk ))
(((student1->mtab)->classObject),
somLId_setUpStudent)))
(student1, "599600", "David Brown"));
```

By adding the name lookup modifier to a method definition, as in this example, we change the underlying method resolution from offset resolution to name-lookup resolution. We would want to make this change in at least one circumstance, when we have the same method name used in two unrelated classes, both of which must be used by the same program. In this case, using default offset resolution will cause a name clash when the macros are defined.

Say, for example, we have a `print()` method defined on both students and dogs, and we have a program that uses both students and dogs. Default resolution will give a name clash when the `print()` macro is defined twice, once for students and once for dogs. One way out of this dilemma is to modify `print()` to use name-lookup resolution, which is compatible with multiple definitions.

DISPATCH RESOLUTION

We have seen that name-lookup resolution can be used when we know which parameters a method will take but don't know the name

```
/* Free argument list.
----- */
for(i=0; i < argNo; ++i){
    arg = *(char **)(argList + (i * sizeof(char *)));
    free(arg);
}
free(argList);

/* Find out what they want to do next time through.
----- */
getline(line, "\n Menu Options are 'm(message)' , 'q(quit)'\n$");
}

/* Clean up.
----- */
_somFree(student);
```

Figure 6. Student class interpreter code (continued from page 99)

```
Menu Options are 'm(message)' , 'q(quit)' : m
    Enter method name: setUpStudent
    Enter the number of arguments: 2
    Enter argument[0]: 100
    Enter argument[1]: David
    Enter the return type: void

Menu Options are 'm(message)' , 'q(quit)': m
    Enter method name: printStudentInfo
    Enter the number of arguments: 0
    Enter the return type: void

Id      : 100
Name    : David
Type    : student

Menu Options are 'm(message)' , 'q(quit)': m
    Enter method name: getStudentType
    Enter the number of arguments: 0
    Enter the return type: string

==> Return string from <getStudentType> method is : <student>.

Menu Options are : 'm(message)' , 'q(quit)' : q
```

Figure 7. Output from typical student interpreter run

of the method until run time. Dispatch resolution takes this one step further; it can be used when we know neither the name, type, or number of parameters the method

will take. This problem might be encountered, for example, when writing an interpreter.

Let's consider writing an interactive interpreter for the student



	Offset	Name-lookup	Dispatch
Resolution Speed	***	**	•
Ease of Use	***	**	•
Method Name	compile time	run time	run time
Parameters	compile time	compile time	run time
*** = most optimized ** = second best optimized • = least optimized			
compile time = Fixed at compile time run time = Can be determined at run time			

Figure 8. Comparison of different resolution strategies

class. At run time, we want to be able to do anything possible with a student by typing in commands to the interpreter. In this case, we cannot use name-lookup resolution because we don't know which method a user might ask to be applied to student. Depending on which method is requested, it may or may not take a parameter. The code for a student interpreter is shown in Figure 6, with the output from a typical run of this program shown in Figure 7.

If we examine the code for the student interpreter, we see exactly how dispatch resolution is used. First, we need to find out the name of the method:

```
getline(method, ">
  Enter method name:\n$");
```

Then we need to find out the number of arguments:

```
getline(line, "> Enter the number
of arguments:\n$");
argNo = atoi(line);
```

Then we prepare an argument list, a two-stage process. First, because all the parameters are string pointers, we allocate enough space for a buffer of pointers. If we had integer arguments, for example, we

would need to make sure the buffer was the correct size to hold integers. (The size of the buffer equals the number of arguments multiplied by the size of a pointer, with the nth element in the buffer containing a pointer to the nth argument.) The code to allocate our buffer is:

```
argList = (char *) malloc((sizeof
(char *)) * argNo);
```

Finally, we allocate space for each argument and set the corresponding pointer in the argument list array:

```
for(i=0; i < argNo; ++i){
  sprintf(msg,
"> Enter argument[%d]:\n$", i);
  arg = (char *) malloc(MAX_LENGTH);
  getline(arg, msg);
  *(char **)(argList +
(i * sizeof(char *))) = arg;
}
```

We are now ready to invoke the method. The code to invoke a method returning a string, for example, looks like:

```
str = (char *)
SOMObject_somDispatchA(student,
somIdFromString(method), "",
argList);
```

(The third argument to SOMObject_som-

DispatchA is a null string. The real argument is what is called a descriptor string, which is not needed for these examples and will not be discussed here.)

Finally, we print out the return value:

```
printf("\n==> Return string from
<%s> method is : <%s>.\n", method,
str);
```

Dispatch resolution is the most flexible of the various dispatch mechanisms, but it is also the slowest.

The important code here is:

```
SOMObject_somDispatchA(student,
somIdFromString(method), "",
argList);
```

which actually dispatches to the appropriate method. This code knows nothing about the class of the target object or the nature of the parameters.



Dispatch resolution is the most flexible of the various dispatch mechanisms, but it is also the slowest and, from a programmer's perspective, the most difficult to use. Beyond the options discussed in this article, it is also possible to

It is also possible to take complete control of the dispatching process by overriding the dispatch methods defined in SOMObject.

take complete control of the dispatching process by overriding the dispatch methods defined in SOMObject. We will save a discussion of this advanced topic for a future article.

SUMMARY

SOM supports three different method resolution strategies: offset, name-lookup, and dispatch. If we compare flexibility, ease of use, and resolution speed, we find that each has a different optimization pattern. This information is summarized in Figure 8.

This ability to choose from three resolution strategies is an important feature of SOM and allows SOM to be used in a variety of different programming situations. At one end of the spectrum, SOM supports the highly efficient but inflexible offset resolution, similar to the virtual mechanism of C++. At the other end is the highly flexible dispatch resolution, which allows methods to be resolved when absolutely no compile time information is available.

ACKNOWLEDGMENTS

SOM is the work of many people. Mike Conner developed the idea and implementation and continues to lead the overall design. Andy

Martin designed and implemented the class interface language and compiler. Larry Raper implemented many features of the run-time library and ported SOM to OS/2. Larry Loucks provided close technical tracking and was instrumental in focusing team effort. The case study used here is based on three frameworks implemented with SOM, the persistence, replication, and record and playback frameworks. Other SOM developers include Scott Danforth, Hari Madduri, Liane Acker, and Kathy Bohrer, led by project managers Cliff Reeves, Rose Ann Roth, and Jerry Ronga.

Nurcan Coskun, IBM Corp., 11400 Burnet Rd., Austin, Texas 78758. Coskun's expertise is in integrated programming environments, code generators, incremental compilers, interpreters, language-based editors, symbolic debuggers, application frameworks, and language design. She has worked on the OS/2 Database Manager and is now working on object-oriented programming environments. She holds a B.S. in industrial engineering from Middle East Technical University and an M.S. and Ph.D. in computer science from the University of Missouri, Rolla. Coskun can be contacted on Internet at nurcan@ausvm1.vnet.ibm.com.

Roger Sessions, IBM Corp., 11400 Burnet Road, Austin, Texas 78758. Sessions is the author of *Reusable Data Structures for C*. His most recent book, *Class Construction in C and C++: Object-Oriented Programming Fundamentals*, was chosen as a main selection for the Library of Computer and Information Science book club. He has authored many articles and frequently lectures on object-oriented programming, C++, and SOM. He is working on persistent SOM frameworks and has previously worked with high-performance relational databases and object-oriented storage systems. Sessions has a B.A. from Bard College and an M.E.S. in database systems from the University of Pennsylvania. He can be contacted on Internet at roger@vnet.ibm.com.

REFERENCES

Coskun, Nurcan, and Roger Sessions. "Object-Oriented Programming in OS/2 2.0," *IBM Personal Systems Developer* (Winter 1992): 107-119. A general introduction to SOM.

Coskun, Nurcan, and Roger Sessions. "Class Objects in SOM," *IBM OS/2 Developer* (Summer 1992): 67-77. A look at the advanced capabilities of SOM.

Sessions, Roger. *Class Construction in C and C++: Object-Oriented Programming Fundamentals*, Englewood Cliffs, N.J.: Prentice-Hall, 1992 (IBM Doc. S242-0086, ISBN 0-13-630104-5).

Orfali, Robert, and Dan Harkey. *Client/Server Programming with OS/2 2.0, Second Edition*. New York: Van Nostrand Reinhold, 1992 (IBM Doc. G325-0650). Includes two basic chapters on SOM.

OS/2 2.0 Technical Library System Object Model Guide and Reference (IBM Doc. S10G-6309)



IBM's System Object Model (SOM) is an object-oriented programming and packaging technology that provides three kinds of method-resolution mechanisms: offset, name-lookup, and dispatch. This article compares offset and name-lookup resolution techniques and shows how to use name-lookup resolution to minimize dependencies between classes. **BY NURCAN COSKUN**

Using Name-Lookup Resolution In SOM

SOM IS A LANGUAGE-NEUTRAL OBJECT model used in OS/2 2.0. SOM provides three kinds of method resolution. Offset resolution, the default for resolving methods in SOM, depends on the class hierarchy in which the method is defined. Name-lookup resolution technique, on the other hand, depends on the method signature only but creates no dependency on the class hierarchy in which the method is defined. The SOM run-time environment includes class objects that can be queried for a method pointer to a given method name. The method pointer obtained can then be used to invoke methods—a type of method resolution not available in static object models such as C++. The third method, dispatch resolution, is the most flexible option and depends on neither class hierarchy nor method signature.

The techniques in this article are discussed in more detail in the article "Method Resolution in SOM," pages 94-102 of this issue.

REDUCING DEPENDENCY BETWEEN CLASSES

When it is necessary to eliminate dependency on the class hierarchy in which a method is defined, name-lookup resolution is the best choice. Figures 1 through 7 detail a sample program that illustrates this concept. The main program, shown in Figure 7, uses the six classes defined in Figures 1 through 6 to show how to reduce dependency between

classes through name-lookup resolution. The complete programs referred to in this article are available in full in the OS2DF2 forum on CompuServe™.

Let us look at the three classes in Figures 1 through 3. The classes in Figures 1 and 2, `NamedObject1` and `FormatObject1`, are not dependent on any other class. On the other hand, the `PrintObject` class in Figure 3 depends on the `NamedObject1` and `FormatObject1` classes. The `PrintObject` class is designed to format objects defined by the `NamedObject1` class, using a format object defined by the `FormatObject1` class. The implementation of the `PrintObject` class uses offset method resolution to send messages to the `NamedObject1` and `FormatObject1` objects.



Nurcan Coskun

When it is necessary to eliminate dependency on the class hierarchy in which a method is defined, name-lookup resolution is the best choice.

The class `PrintObjectImproved`, defined in Figure 4, is an improved version of the `PrintObject` class. It can format objects that belong to different classes using different formatters.



The signatures of the methods supported by these different objects before they can be passed to the `PrintObjectImproved` methods are documented in the class definition file. The class implementation file for `PrintObjectImproved` uses SOM

The `PrintObjectImproved` class can work with objects of multiple classes as long as the objects support the required method signature.

name-lookup resolution to send messages to the objects. In other words, the `PrintObjectImproved` class can work with objects of multiple classes as long as the objects support the required method signature.

In the main program, shown in Figure 7, the `PrintObjectImproved` class is used with `NamedObject1` and `FormatObject1` as well as `NamedObject2` and `FormatObject2`.

Name-lookup resolution technique is used here to reduce the dependencies between classes. Another way to reduce class dependencies is to use abstract base classes. For example, we could create two abstract classes, one for the named object and another for the format object. Our improved print object could be implemented with these abstract classes, while different versions of named and format objects could be created by subclassing the corresponding abstract class. (The abstract class solution requires the use of offset resolution and will be discussed in a future article.) It is best to use abstract classes if there is much dependency

```
/* Class Definition File(namedo1.csc) */

include <somobj.sc>

class:
    NamedObject1;

parent:
    SOMObject;

data:
    char name[32];

methods:

    void setName(char *name);
    char *getName();

-----

/* Class Implementation File(namedo1.c) */

#define NamedObject1_Class_Source
#include "namedo1.ih"

SOM_Scope void SOMLINK setName(NamedObject1 *somSelf,
                                char *name)
{
    NamedObject1Data *somThis = NamedObject1GetData(somSelf);
    strcpy(_name, name);
}

SOM_Scope char * SOMLINK getName(NamedObject1 *somSelf)
{
    NamedObject1Data *somThis = NamedObject1GetData(somSelf);
    return _name;
}
```

Figure 1. Class definition and implementation files for `NamedObject1`

between classes. On the other hand, if there are few class dependencies, name resolution is more advantageous because it requires no abstract class definition.

Frameworks. A framework is a collection of classes designed to solve a specific problem. Framework technology increases code reusabil-

ity, shortening the product development cycle. Frameworks must be designed carefully, however, to maximize code reuse. If the framework is designed to depend heavily on other frameworks, code reuse will be minimal, as a dependent framework can be used only with the frameworks on which it is dependent. It cannot use alterna-



```
/* Class Definition File(formato1.csc) */

include <somobj.sc>

class:
  FormatObject1;

parent:
  SOMObject;

methods:

  void formatName(char *text, char *buffer);
  -- Formats an ascii data stream.

-----

/* Class Implementation File(formato1.c) */

#define FormatObject1_Class_Source
#include "formato1.ih"
#include <ctype.h>

SOM_Scope void  SOMLINK formatName(FormatObject1 *somSelf,
                                   char *text, char *buffer)
{
  int i;
  /* Convert the text to upper case */
  for(i=0; i<strlen(text); i++){
    buffer[i] = toupper(text[i]);
  }
  buffer[i] = '\0';
}
```

Figure 2. Class definition and implementation files for FormatObject1

tive frameworks even if they provide similar functionality.

The technique in the following case study takes advantage of SOM's name-lookup scheme for method resolution, which minimizes the dependency between frameworks.

A CASE STUDY: PERSISTENT FRAMEWORK-INDEPENDENT RECORD AND PLAYBACK FRAMEWORK

An application framework is a collection of frameworks that simplifies

application development through class libraries or frameworks that provide reusable code. The developer uses the code, adding new features through specialization of the framework classes.

The record and playback framework, one of the most useful application frameworks, lets a developer record and play back method calls. This type of function can be used for undo, record and playback, extension language support, persistence, and replication.

In a replication framework, one

requirement is to create an external representation for method calls that can be transferred to a remote host or another process. The receiver of the external representation then reconstructs the method call from the external representation and executes the appropriate method procedure. For this to be possible, all information about method calls, such as the target object, method name, and arguments, should be able to be externalized.

Externalizing method names and primitive data types such as

In a replication framework, one of the requirements is to create an external representation for method calls that can be transferred to a remote host or another process.

integer, real, character, and string names is very straightforward. A more complex data type is the persistent object data type, which has persistent IDs that can be retrieved from the persistence framework, which is then used by the application framework. Given such an ID, it is possible to request an object from the persistence framework. In other words, the record and playback framework needs to communicate with the persistence framework to externalize and internalize the object arguments. This type of communication can be performed independently of the persistence framework. The application programming interfaces in Figure 8 are designed to provide this type of


```

/* Class Definition File(printo.csc) */

include <somobj.sc>

class:
    PrintObject;

parent:
    SOMObject;

passthru: C.h, after;
    #include "formato1.h"
    #include "namedo1.h"
endpassthru;

data:
    FormatObject1 *formatter;

methods:

    override somInit;
    void      printData(NamedObject1 *object);

-----

/* Class Implementation File(printo.c) */

#define PrintObject_Class_Source
#include "printo.ih"

SOM_Scope void  SOMLINK somInit(PrintObject *somSelf)
{
    PrintObjectData *somThis = PrintObjectGetData(somSelf);
    _formatter = FormatObject1New();
    parent_somInit(somSelf);
}

SOM_Scope void  SOMLINK printData(PrintObject *somSelf,
    NamedObject1 *object)
{
    PrintObjectData *somThis = PrintObjectGetData(somSelf);
    char *data;
    char  buffer[32];
    data = _getName(object);
    _formatName(_formatter, data, buffer);
    printf("Formatted form for <%s> is <%s>\n", data, buffer);
}

```

Figure 3. Class definition and implementation files for PrintObject



```
/* Class Definition File(printoi.csc) */

include <somobj.sc>

class:
  PrintObjectImproved;

parent:
  SOMObject;

data:
  SOMObject *formatter;

methods:

  override    somInit;

  void        setFormatter(SOMObject *formatter);
  -- Parameter 'formatter' must support a method with
  -- this signature:
  --   typedef void formatDataTD (SOMObject *, char *, char *);

  SOMObject *getFormatter();

  void        printObjectData(SOMObject *object,
                              char *getDataMethodName, char *formatDataMethodName);
  -- Parameter 'object' must support a method with this
  -- signature:
  --   typedef char * getDataTD (SOMObject *);

-----

/* Class Implementation File(printoi.c) */

#define PrintObjectImproved_Class_Source
#include "printoi.ih"

typedef char * getDataTD (SOMObject *);
typedef void  formatDataTD (SOMObject *, char *, char *);

SOM_Scope void  SOMLINK somInit(PrintObjectImproved *somSelf)
{
  PrintObjectImprovedData *somThis = PrintObjectImprovedGetData(somSelf);
  _formatter = NULL;
  parent_somInit(somSelf);
}

SOM_Scope void  SOMLINK setFormatter(PrintObjectImproved *somSelf,
```

Figure 4. Class definition and implementation files for `PrintObjectImproved` (continued on page 108)


```

        SOMObject *formatter)
{
    PrintObjectImprovedData *somThis = PrintObjectImprovedGetData(somSelf);
    _formatter = formatter;
}

SOM_Scope SOMObject * SOMLINK getFormatter(PrintObjectImproved *somSelf)
{
    PrintObjectImprovedData *somThis = PrintObjectImprovedGetData(somSelf);
    return (_formatter);
}

SOM_Scope void SOMLINK printObjectData(PrintObjectImproved *somSelf,
        SOMObject *object,
        char *getDataMethodName,
        char *formatDataMethodName)
{
    PrintObjectImprovedData *somThis = PrintObjectImprovedGetData(somSelf);
    char *data;
    char buffer[32];
    somId methodId;
    somMethodProc *m;

    methodId = somIdFromString(getDataMethodName);
    _somFindMethodOk(SOM_GetClass(object), methodId, &m);
    data = ((getDataTD *)m) (object);

    methodId = somIdFromString(formatDataMethodName);
    _somFindMethodOk(SOM_GetClass(_formatter), methodId, &m);
    ((formatDataTD *)m) (_formatter, data, buffer);

    printf("Formatted form for <%s> is <%s>\n", data, buffer);
}

```

Figure 4. Class definition and implementation files for `PrintObjectImproved` (continued from page 107)

function.

The `somExternalizeCommand` interface creates an external representation for a given method call, which is returned in a buffer. As an example, suppose we would like to externalize the target object, method name, class name, and a variable number of arguments. The first parameter is the name of the method used to get the object ID from a SOM object. If the method named by the `idFromObject-`

Method parameter is supported by the target object, the corresponding object is persistent and we can send the message named by the `idFromObjectMethod` parameter to the object, retrieving its object ID.

The `somCreateCommand` interface uses the external representation created by `somExternalizeCommand` to construct a command object, which can then be triggered to execute the method call externalized. Within

the external representation of a method call, the objects are represented with their object IDs. Given such an ID, a persistence framework-supported name space manager returns the corresponding object.

In other words, persistence frameworks have interfaces that return the objects for a given persistent object ID. The name of the method needed to get an object



```
/* Class Definition File(namedo2.csc) */

include <somobj.sc>

class:
    NamedObject2;

parent:
    SOMObject;

data:
    char objectData[32];

methods:

    void setObjectData(char *objectData);
    char *getObjectData();

-----

/* Class Implementation File(namedo2.c) */

#define NamedObject2_Class_Source
#include "namedo2.i.h"

SOM_Scope void    SOMLINK setObjectData(NamedObject2 *somSelf,
                                         char *objectData)
{
    NamedObject2Data *somThis = NamedObject2GetData(somSelf);
    strcpy(_objectData, objectData);
}

SOM_Scope char * SOMLINK getObjectData(NamedObject2 *somSelf)
{
    NamedObject2Data *somThis = NamedObject2GetData(somSelf);
    return (_objectData);
}
```

Figure 5. Class definition and implementation files for NamedObject2

from an ID, plus a pointer to the name space manager, are passed as parameters to the `somCreateCommand` interface (the `objectFromIdMethod` and `nameSpaceManager` parameters). The method, named by the `objectFromIdMethod` parameter, should be supported by the name space manager, identified by the `nameSpaceManager`

parameter. The record and playback framework then uses SOM to get a pointer to a procedure that implements the `objectFromIdMethod` method. The method procedure pointer is then used to send a message to the name space manager with an argument of persistent object ID. Finally, the name space

manager returns the corresponding object.

Using static offset method resolution to invoke methods of the persistence framework can cause a strong and undesirable coupling between the frameworks. With the name-lookup resolution technique, however, a framework-indepen-


```

/* Class Definition File(formato2.csc) */

include <somobj.sc>

class:
    FormatObject2;

parent:
    SOMObject;

methods:

    void formatObjectData(char *text, char *buffer);
    -- Format ascii data stream.

-----

/* Class Implementation File(formato2.c) */

#define FormatObject2_Class_Source
#include "formato2.ih"
#include <ctype.h>

SOM_Scope void    SOMLINK formatObjectData(FormatObject2 *somSelf,
char *text, char *buffer)
{
    int i;
    /* Convert the text to lower case */
    for(i=0; i<strlen(text); i++){
        buffer[i] = tolower(text[i]);
    }
    buffer[i] = '\0';
}

```

Figure 6. Class definition and implementation files for FormatObject2

dent record and playback framework can be designed to document the signature of any persistence framework interfaces. Much weaker coupling is therefore achieved with dynamic name-lookup method resolution.

SUMMARY

The name-lookup method resolution technique can reduce depen-

dependency between classes, which can document their mutual dependencies by publishing the signatures of the interfaces they use.

This concept has several advantages. The static dependency between the related frameworks is minimized, while the frameworks may be recompiled only when the code needs to be changed (for example, when the interface signa-

ture changes). The framework dependencies can be clearly documented by publishing the necessary interface signatures. The only concrete design decision required is choosing the signature of the interface. This information can be used by other vendors to develop new frameworks that work with existing ones.



Main Program:

```
-----

#include "namedo1.h"
#include "formato1.h"
#include "printo.h"
#include "printo1.h"
#include "namedo2.h"
#include "formato2.h"

void main(){
    NamedObject1 *namedo1 = NamedObject1New();
    FormatObject1 *formato1 = FormatObject1New();
    PrintObject *printo = PrintObjectNew();
    PrintObjectImproved *printo1 = PrintObjectImprovedNew();
    NamedObject2 *namedo2 = NamedObject2New();
    FormatObject2 *formato2 = FormatObject2New();

    /* <PrintObject> objects can only be used with */
    /* <NamedObject1> and <FormatObject1> objects */
    _setName(namedo1, "Object Data One");
    _printData(printo, namedo1);

    /* use <PrintObjectImproved> objects with */
    /* <NamedObject1> and <FormatObject1> objects */
    _setName(namedo1, "Object Data Two");
    _setFormatter(printo1, formato1);
    _printObjectData(printo1, namedo1, "getName", "formatName");

    /* use <PrintObjectImproved> objects with */
    /* <NamedObject2> and <FormatObject2> objects */
    _setObjectData(namedo2, "Object Data Three");
    _setFormatter(printo1, formato2);
    _printObjectData(printo1, namedo2, "getObjectData", "formatObjectData");
}
```

Output:

```
-----
Formatted form for <Object Data One> is <OBJECT DATA ONE>
Formatted form for <Object Data Two> is <OBJECT DATA TWO>
Formatted form for <Object Data Three> is <object data three>
```

Figure 7. Client code and output



ACKNOWLEDGMENTS

SOM is the work of many people. Mike Conner developed the idea and implementation and continues to lead the overall design. Andy Martin designed and implemented the class interface language and compiler. Larry Raper implemented many features of the run-time library and ported SOM to OS/2. Larry Loucks provided close technical tracking and was instrumental in focusing team effort. The case study used here is based on three frameworks implemented with SOM, the persistence, replication, and record and playback frameworks. Other SOM developers include Roger Sessions, Scott Danforth, Hari Madhuri, Liane Acker, and Kathy Bohrer, led by project managers Cliff Reeves, Rose Ann Roth, and Jerry Ronga.

Nurcan Coskun, IBM Corp., 11400 Burnet Rd., Austin, Texas 78758. Coskun's expertise is in integrated programming environments, code generators, incremental compilers, interpreters, language-based editors, symbolic debuggers, application frameworks, and language design. She has worked on the OS/2 Database Manager and is now working on object-oriented programming environments. She holds a B.S. in industrial engineering from Middle East Technical University and an M.S. and Ph.D. in computer science from the University of Missouri, Rolla. Coskun can be contacted on Internet at nurcan@ausvm1.vnet.ibm.com.

```
somExternalizeCommand(char *idFromObjectMethod,
                      char **buffer, int *length, SOMObject *target,
                      char *className, char *method, ...);

somCreateCommand(char *objectFromIdMethod,
                 char *buffer, int length, SOMObject *nameSpaceManager);
```

Figure 8. Record and playback framework APIs

REFERENCES

Coskun, Nurcan, and Roger Sessions. "Object-Oriented Programming in OS/2 2.0," *IBM Personal Systems Developer* (Winter 1992): 107-119. A general introduction to SOM.

Coskun, Nurcan, and Roger Sessions. "Class Objects in SOM," *IBM OS/2 Developer* (Summer 1992): 67-77. A look at the advanced capabilities of SOM.

Coskun, Nurcan, and Roger Sessions. "Method Resolution in SOM," *IBM OS/2 Developer* (Spring 1993): 94-102.

Orfali, Robert, and Dan Harkey. *Client/Server Programming with OS/2 2.0, Second Edition*. New York: Van Nostrand Reinhold, 1992 (IBM Doc. G325-0650). Includes two basic chapters on SOM.

OS/2 2.0 Technical Library System Object Model Guide and Reference (IBM Doc. S10G-6309)





IBM's LAN NetView Start™ product (Start) brings one-stop configuration and management to workstations, providing users with a powerful graphical user interface that allows users to define, configure, and update workstations in a network. **BY THEODORE SHRADER and KHALIL EMAMI**

Network Plotting Formation: The LAN NetView Start Objects

IBM's LAN NETVIEW™ START PRODUCT (Start) brings one-stop configuration and management to workstations, providing users with a powerful graphical user interface that allows users to define, configure, and update workstations in a network. It takes advantage of OS/2 2.0's Workplace Shell, letting users drag and drop icon objects to create workstation nodes and other objects and draw connecting relationships between them.

Start supports the configuration and installation of the following operating system and subsystem products on a workstation:

- OS/2 2.0 (through the use of supplemental response files)
- Extended Services™ 1.0 for OS/2 (including the Database Manager and Communication Manager products)
- LAN Server™ 3.0 Entry or Advanced
- Network Transport Services/2™ (the LAN Adapter and Protocol Support components).

Start supports these products by creating product response files, LAN CID Utility command files, or NetView Distribution Manager/2 (NetView DM/2) change files that can be used to install and configure the desired products on a target workstation. (The LAN CID Utility is a component of the Network Transport Services/2™ product.) The response files

provide the same information to the configuration program as if a user is physically at the workstation inputting those values. LAN CID Utility command files drive the remote installation and configuration of the target workstation. This "lightly attended" installation allows users to start the command-file sequence at the target workstation, answer a few questions, and leave without waiting until the installation is complete. NetView DM/2 uses the change files produced by Start to install and update workstations.

But users are not limited to installing LAN, relational database, and communications support on their workstations. Applications enabled for configuration, installation, and distribution (CID) are also supported. Start generates LAN CID Utility command files or NetView DM/2 change files in conjunction with response files previously generated for CID-enabled and non-CID-enabled applications, which helps automate the installation and configuration of these products. Start also supports a variety of adapters, including token ring and many of the available Ethernet adapters.

Although response files are generated only for workstations configured with OS/2 2.0, Start does not prevent users from representing network workstation nodes that run with OS/2 1.3, DOS, or DOS with Windows. In fact, the use of heterogeneous workstations are an integral part of the network environ-



Theodore Shrader



Khalil Emami

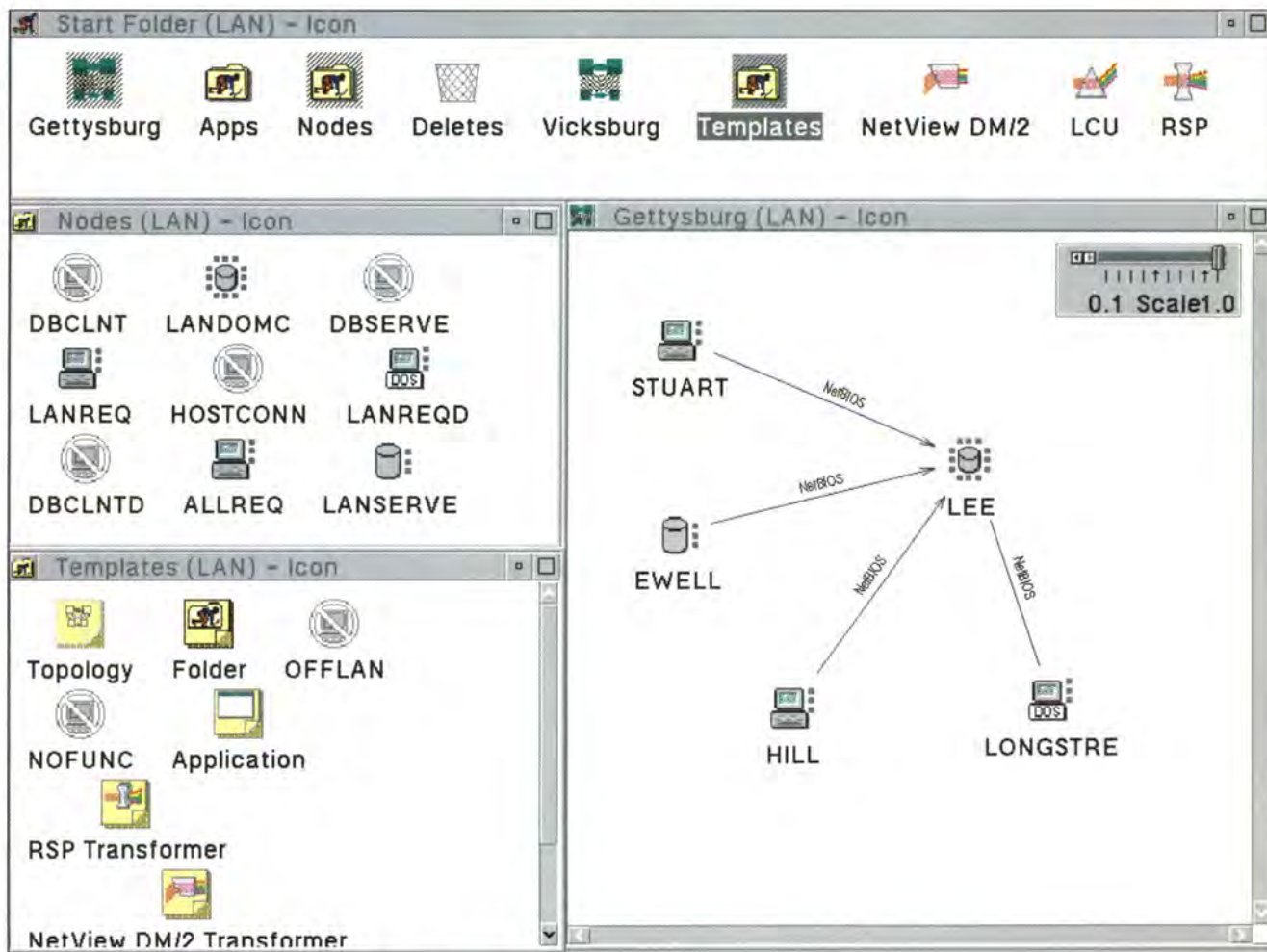


Figure 1. Topology in LAN view, flanked by the Nodes and Template folders

ment. For example, some values in the generated response files are calculated based on the number of DOS workstations present in the network.

class managers, and presented object information through view classes. Start users can approach network creation and maintenance by creating and configuring their

used to store their data values.

LAN NETVIEW START

Data objects. Start handles its objects in a tree structure. The network object is at the top root of the tree, with folders and topologies beneath it and nodes contained in the topologies. In contrast to folders, which are container objects that allow storage of real and template objects, topologies allow users to subdivide workstations into different groups rather than fitting all workstations into a single container object. Start does not impose a rigid hierarchy; the program also uses other objects that can be stored within one another. Application objects, for example, can be stored within a workstation node. Topolo-

Start handles its objects in a tree structure. The network object is at the top root of the tree, with folders and topologies beneath it and nodes contained in the topologies.

Written in Smalltalk/V™ for OS/2, Start was designed and developed as an object-oriented application. Just as Start developers created object classes, established

network objects in an object-oriented fashion. This article discusses the key objects used by Start and gives an in-depth examination of the ASCII and SQL database managers

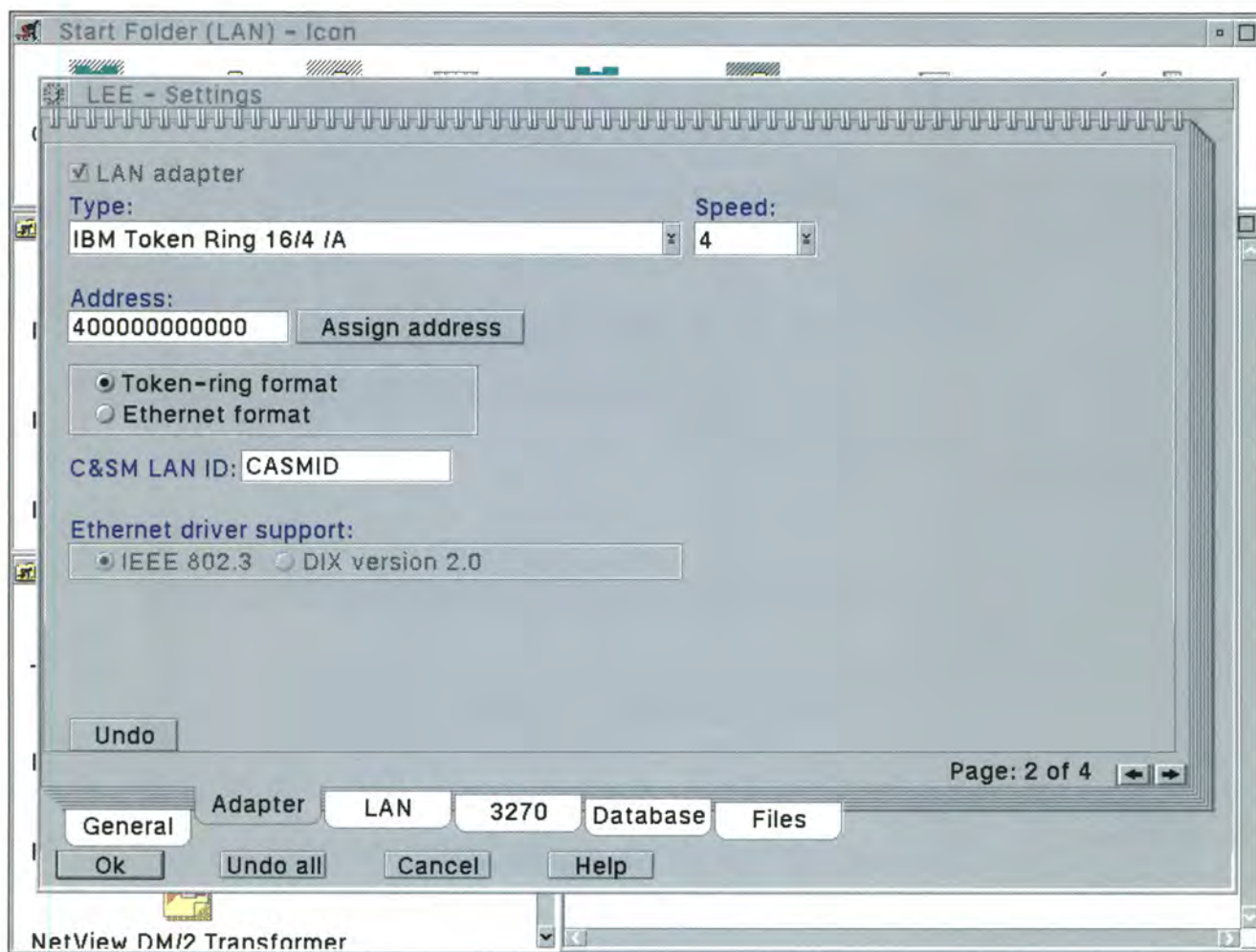


Figure 2. Second page of the adapter section in the node notebook

gies can also contain other topologies, while folders can contain topologies and the unique transformer objects. The available objects within the Start program are topologies, nodes, transformers, applications, and folders.

Topologies. A topology is the user's work area. Topologies, of which many can exist, portray workstation nodes and the connections between them for groups such as a single department or a building floor. Within each topology are three supported connection views: LAN, 3270 emulation, and database. Figure 1 shows a topology in the LAN view. In Figure 1, the zoom slider control allows more of the topology to be seen in the window. The nodes folder shows the

default templates for eight different types of workstations and software configurations, from a DOS LAN Requester to a database server. The nodes folder also includes a template object representing a host machine. Users can create additional node templates. The template folder supplies templates to create topologies, folders, applications, and transformers.

Only one connection view is shown at a time, and it highlights the nodes and connections within the topology that are defined with the functions relating to that connection view. For example, the LAN view will show all the nodes defined with OS/2 LAN Requester, OS/2 LAN Server (entry or advanced), DOS LAN Requester, or domain controller functions. Con-

nections from the requesters and servers to the domain controllers are displayed along with the connection type (such as NetBIOS). Nodes that do not have LAN function are shown as greyed workstations with a NOT symbol. Database and 3270 connections are not displayed. (The icons for nodes with multiple functions change according to the view in which they are shown; a node defined with LAN Requester and database server functions, for example, will have a requester icon in the LAN view, a greyed icon in the 3270 view, and a cylindrical database server icon in the Database view.)

The user can draw connections between nodes only in the active topology view. For example, connections between database clients

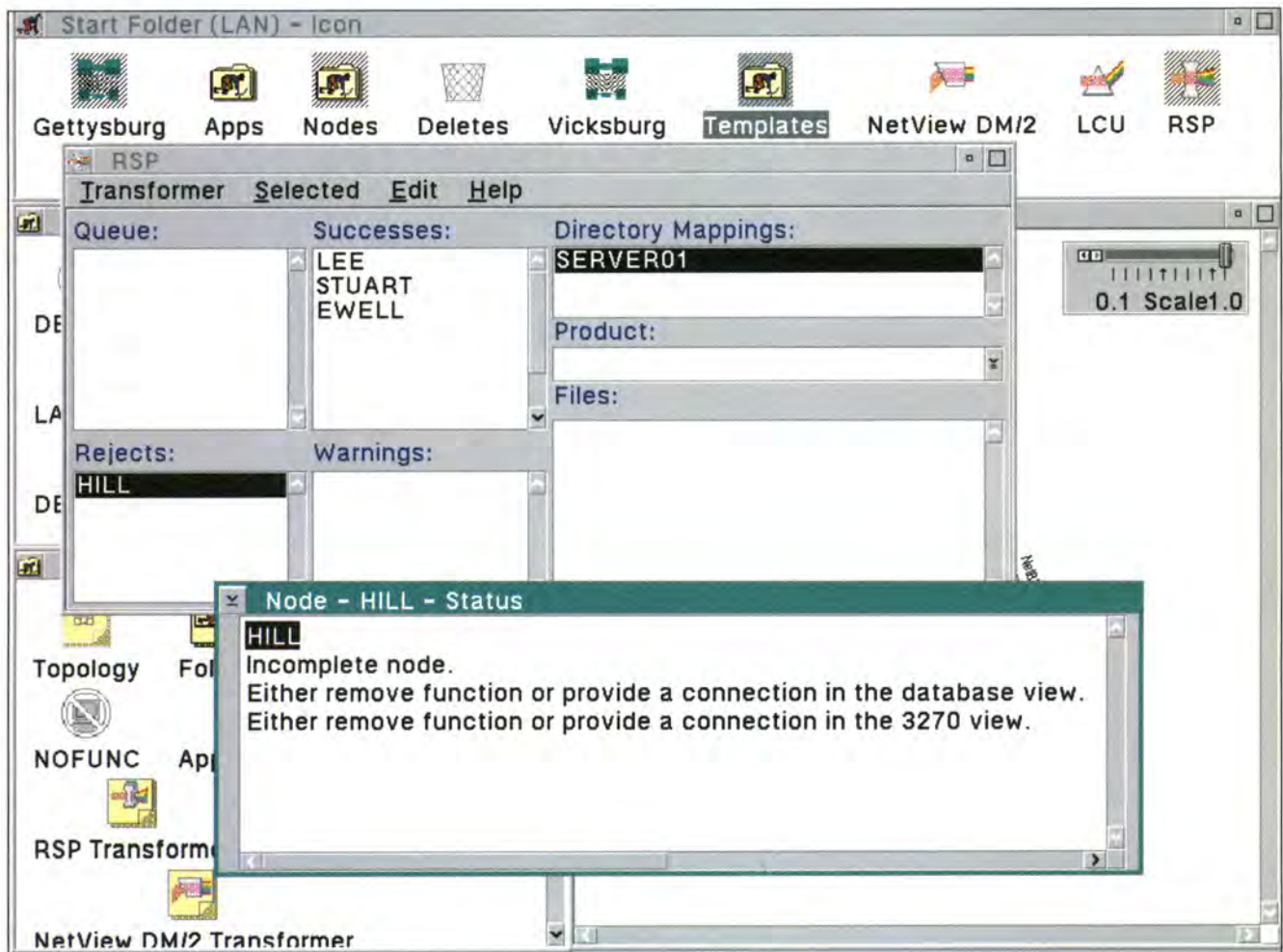


Figure 3. RSP Transformer window with the rejected nodes reason dialogue box

and servers can be drawn only in the Database view. To draw connections, the user holds down the first mouse button on the source, drags the mouse pointer to the target, and releases the button. Valid connections are drawn; for those that are invalid, Start posts a message box that explains why. For example, one box explains that the maximum number of supported clients for a database server has been reached.

The 3270 view allows DFT and non-DFT (token-ring and SDLC) connections between nodes equipped with 3270 emulation and host nodes, which are representations of host machines. Start does not generate response files for these host nodes; they exist to allow users to define 3270 connections for other nodes with

3270 emulation. Once defined, the nodes with 3270 sessions can have response files generated for them.

The three topology connection views help organize information for the user. All nodes in a topology are present in these views, yet by switching between connection views, the user can immediately see and manipulate those nodes and connections relating to the active view.

Nodes. Nodes are the next most basic object after their parent topologies. Except for host nodes, all nodes represent workstations. Users can define the attributes and functions of a node by double-clicking on it to open the node's settings notebook. Figure 2 shows the second page of the adapter section in a

node settings notebook. All Start objects have a modifiable settings notebook. Many, like the topology, folder, and node objects, also have details and icon presentation views. A node's open icon view shows the applications targeted for installation on the node; a topology details view is a table of all the nodes in the topology, their attributes, and their connections.

The node notebook is divided into six sections.

The *General* section contains workstation hardware, SNA information, and inventory data. Here, the user can define values, for instance, renaming a node's local alias name by changing the value in the corresponding entry field or switching a node's operating system from OS/2 1.3 to OS/2 2.0 by

selecting a different entry from the Operating System pull-down selection list.

The *Adapter* section allows the user to define an adapter such as DFT, SDLC, token ring, or Ethernet. Its multiple definition pages allow the user to specify values such as adapter type, speed, addresses, and number of other 802.2 applications. Ethernet adapters have additional entry fields that become active if an Ethernet adapter is chosen from the selection list.

The *LAN* section lets users specify the LAN function, if any, and allows them to customize settings, such as the percentage of concurrent OS/2 and DOS requesters, the file access mode, and use of the LAN messenger service.

The *3270* section has a radio button to indicate whether the node has 3270 emulator function. If this function is selected, up to five DFT and five non-DFT sessions can be specified along with parameters such as their device, session ID, or short ID.

The *Database* section, similar to the LAN and 3270 sections, gives the user a choice of database functions for the node, such as database server or no database function, and provides entry fields for values such as the database workstation name.

The *Files* section stores and presents a list of supplementary response files for the operating system and subsystem products that can be specified with Start, along with the target drive for installation. Supplemental response files allow users to add specific configuration values to the generated response files.

Transformers. There are three types of transformers: RSP (response), NetView DM/2, and LCU (LAN CID Utility). The RSP transformer creates response files for nodes dragged and dropped onto it. If a window isn't already open, the

transformer creates one that gives users status information about each node, as shown in Figure 3. The nodes are placed in a queue, with a dialogue box indicating the transformer's progress. Nodes successfully transformed by Start are logged on a Successes list. Successful nodes, still present on the topology, now have associated response files stored in the response file directory hierarchy. Rejected nodes are banished to the Rejects list. (Nodes can be rejected if they have

products on the target workstation. With the LCU transformer, users can automate almost the entire process of workstation installation and configuration and edit all response and command files created by Start. A sample response file is shown in Figure 4.

Applications. CID-enabled applications in the Start realm can be represented by icon objects, and have user-modifiable fields for the product name, install program, applica-

A node can be successful and still have encountered a warning; for example, if a calculated value exceeding a maximum range was to default to the maximum value.

invalid values that cannot be reconciled, or if their connections are not completely defined.) A node can be successful and still have encountered a warning; for example, if a calculated value exceeding a maximum range was to default to the maximum value. These nodes are placed in the Warnings list. Users can view the directories, products, and files associated with nodes from the RSP transformer window.

The NetView DM/2 transformer works like the RSP transformer in that it creates change files for appropriate subsystems, which are used by NetView DM/2 to install and update the target workstations. Start adds the workstation to the NetView DM/2 catalogue and performs a build on the general change file.

The LCU transformer contains a superset of the function provided by the RSP transformer. It creates product-specific response files as needed and generates the LAN CID Utility command files as required to install and configure

tion resource requirements, and other definable parameter fields. The application install routine uses the generated response file to install and configure the product on the target workstation. Applications can be installed on nodes in an object-oriented manner. They are dragged from their container and dropped onto a node icon or node icon view window. The application can be viewed in the icon view of the node. These applications do not include OS/2 subsystem applications such as Communications Manager that are already integrated into Start.

Folders. Folders act like general-purpose container object windows. The top-level Start folder initially contains a topology, an applications folder, a templates folder, a node templates folder, transformer icons, and the delete wastebasket. (Deleted objects are placed into the wastebasket, from which they can be retrieved unless they have been permanently shredded.) Users can





create folders to store groups of objects such as additional user-defined node templates. Figure 1 also shows the top-level folder.

CREATING A NETWORK

One way to create a new network is to open up a topology, drag and drop nodes from the node templates topology, modify node values if necessary, and draw connections. Start takes much of the book-keeping out of network creation by supplying user-modifiable automatic node and address naming algorithms. Many rules required to create a successful network are built into Start, such as those that insure node name uniqueness to the network and prevent invalid connections.

Users can create new objects in Start just as they would in the OS/2 2.0 environment. Dragging and dropping a template object into a container object (for example, putting a node template into a topology container) creates a new object. Users can modify the default templates to create custom templates that can be shared across multiple nodes; for example, a user might create a template defined with LAN requester and database client functions but not 3270 emulation.

For existing networks, Start provides a migration utility, OCUNODE.EXE, that can be run on each workstation. OCUNODE.EXE creates an import file with much of the information needed by Start to define a node. DFT and non-DFT sessions are included along with the adapter used by the node and LAN parameter information, such as the domain controller name to which the node was connected.

These import files can be brought into a topology object by dragging and dropping an OS/2 file folder containing the import file icons or the individual file icons onto the specified topology window. A successfully migrated

```
; 1-23-1993 4:26:23 pm E:\RSP\LS\OCURSP\NET1\EWELL.RSP
```

```
UPDATEIBMLAN = REQUESTER
<
COMPUTERNAME = NODE0004
DOMAIN = DOMAIN01
>
```

Figure 4. Sample response file for a LAN Requester node for the LAN Services product

```
.NETWORKS
.TOPOLOGIES
18, 2, 2, "Gettysburg", 0, 197640, "NODE****", "0006", "emlan",....
.FOLDERS
.NODES
150, 18, 18, 24, 0, "EWELL", 1, 332, 8, "SERVER01", 0, 0, 16, 1,...
332, 18, 0, 150, 100, "EWELL", 1, 0, 24, "SERVER01", 0, 0, 16, 1,...
32, 18, 18, 25, 0, "LEE", 1, 370, 528392, "SERVER01", 0, 0, 16, 2,...
370, 18, 0, 32, 100, "LEE", 1, 0, 24, "SERVER01", 0, 0, 16, 2, 32,...
268, 18, 18, 29, 0, "LONGSTRE", 1, 0, 8, "SERVER01", 0, 0, 16, 8,...
91, 18, 18, 23, 0, "STUART", 1, 408, 8, "SERVER01", 0, 0, 16, 4,...
408, 18, 0, 91, 100, "STUART", 1, 0, 24, "SERVER01", 0, 0, 16, 4,...
209, 18, 18, 28, 0, "HILL", 1, 0, 8, "SERVER01", 0, 0, 2, 4, 0, ...
.CONNECTIONS
327, 18, 18, 91, 32, "lan", 8, 0, 0, 4, "", 0
328, 18, 18, 150, 32, "lan", 8, 0, 0, 4, "", 0
329, 18, 18, 209, 32, "lan", 8, 0, 0, 4, "", 0
330, 18, 18, 268, 32, "lan", 8, 0, 0, 4, "", 0
.ADAPTERPARMS
333, 332, 0, 156, 100, 0, 5, 8, 4, "400000000002", "T", 0, 0, 0,...
156, 150, 18, 155, 0, 0, 5, 8, 4, "400000000002", "T", 0, 0, 0,...
372, 370, 0, 34, 100, 0, 5, 8, 4, "400000000000", "T", 0, 0, 0,...
34, 32, 18, 33, 0, 0, 5, 8, 4, "400000000000", "T", 0, 0, 0, 0, 0,...
274, 268, 18, 273, 0, 0, 5, 8, 4, "400000000004", "T", 0, 0, 0, 0,...
410, 408, 0, 97, 100, 0, 5, 8, 4, "400000000001", "T", 0, 0, 0, 0,...
97, 91, 18, 96, 0, 0, 5, 8, 4, "400000000001", "T", 0, 0, 0, 0, 0,...
211, 209, 18, 210, 0, 0, 5, 8, 4, "400000000003", "T", 0, 0, 0, 0,...
.EMDFTPARMS
.EMNONDFTPARMS
259, 209, 18, 258, 0, 1, 520, "F", 1, "F", "02"
261, 209, 18, 260, 0, 2, 520, "G", 1, "G", "03"
.INCLUDES
.HOSTPARMS
.APPLICATIONS
.RTRANSFORMERS
.LCUTTRANSFORMERS
.NVTRANSFORMERS
.LCUMAP
```

Figure 5. An ASCII topology file



workstation will appear as an icon in the topology, and connections will automatically be drawn to valid host nodes or domain controllers already present in the topology. (Host node representations are created for the migrated node if necessary.) Start tries to migrate as much information as possible. If some values, such as the import file containing information on an unsupported adapter, are invalid, the adapter information is not included with the node. All defaulted values and parameters that were not migrated are listed in a status dialogue and log file.

Users can modify several workstations at one time by creating an **Attribute Value** file, which contains tags and values for updating workstations whose names are included in the file, and dropping it onto the desired topology. The application viewer also allows users to change application attributes across multiple nodes.

EXAMINING ASCII FILE AND DATABASE MANAGER STORAGE

With Start, users can store network and topology data in ASCII files or in a Database Manager (DBM) relational database. ASCII files take up less disk space, do not require the DBM to be installed, and are easily transportable from one workstation to another. DBM databases have the advantage of being relational, which also allows data to be queried easily by SQL statements or by graphical user interfaces to the DBM such as Query Manager.

Start ASCII files are stored in a single directory with an entire network as its scope; different networks are stored in different directories. The same is true of a DBM database; each database represents a single network, and all information contained in that network is stored in tables in the database. The ASCII and DBM databases, however, diverge in how they store data. For ASCII, the contents of network

and topology objects are stored in separate files. (Folder objects are stored in single files too, but they are considered topology files.) For the DBM, this information is spread across tables. There are fifteen SQL tables. All topology object information, for example, is stored in the **TOPOLOGIES** table. (Start 1.1 adds the **RECYCLE** table to store the set of reusable object IDs.)

TRANSFERRING A NETWORK FROM ASCII TO SQL

Users can easily move a network and its topologies from ASCII files to a DBM database. Each network or topology file has a specific, com-

mon structure. Network filenames have their name along with a .NET extension, while topologies have their object ID number along with a .TOP extension. Object IDs uniquely identify objects in a Start network and are used as primary keys in each of the SQL tables. Except for the data lines after set names, each data row in an ASCII file starts with the ID of that object. Figure 5 shows a sample ASCII topology file.

By dividing each ASCII network and topology file into its respective table files and combining the table files with the same name into a single file, users can create fifteen import files, one for each table.

A sixteenth import file can transfer the contents of the **RECYCLE_ID_SET** in the network file into the **RECYCLE** table of the SQL database. This is only necessary if the user wants to

Unlike topologies, network files have only the network table, with one data row entry followed by several name sets that exist to help guarantee name and address uniqueness when the ASCII database is used.

reuse the object IDs of deleted objects or temporary objects that Start had used. Before the files are imported into a DBM database, the tables and database must be created. To create them, start with the **/DBSQL** and **/CREATEDATABASE** options and exit the program. The database will be primed with the tables needed by the DBM. Figure 6 shows the REXX command file **ASCTOSQL.COMD**, which takes the network and topology files from a specified directory, combines their like data rows into delimited ASCII files, and imports these files into a specified database. This program works with Start databases versions 1.0 and 1.1.

The command file first registers the REXX utilities and initializes the table and set name arrays. Existing delimited ASCII files and

mon structure. Network filenames have their name along with a .NET extension, while topologies have their object ID number along with a .TOP extension. Object IDs uniquely identify objects in a Start network and are used as primary keys in each of the SQL tables. Except for the data lines after set names, each data row in an ASCII file starts with the ID of that object. Figure 5 shows a sample ASCII topology file.

All the SQL table names in Figure 5 are listed with periods at the beginning. (Data rows that exceeded the figure width are truncated with the ... characters.) Following the table name are the data rows for that table. Topologies may have no data rows for some tables and many rows for others.

Unlike topologies, network files



message files are erased, if they exist, before the list of topology and network files are extracted and stored for future reference. Next, the program loops through each of the files and stores the data rows from the *.TOP and *.NET files into the appropriate delimited ASCII files. For example, data lines after the EMDFTPARGS table line are stored in the EMDFTPARG.DEL file. Data lines after the name sets in the network file are ignored.

The delimited ASCII files must be closed before they can be used. The program then connects to the database and loops through each table, deleting any existing rows in the table before importing the appropriate delimited ASCII file. This delete command is done even though the IMPORT replaces the data rows in a table, since the imported file may not contain any data. Finally, the program disconnects from the database.

Some data rows will be rejected, and the *.MSG files will contain informational and warning messages regarding the table imports. To avoid importing a row with the same object ID into the TOPOLOGIES or FOLDERS table, the program ignores the data row in the file with the same object ID as the filename (except the root and pool topology files, which have no direct parent objects). The object ID field is always the first value in the data row. Each data row, hence each object, has a unique object ID. (Only data rows under HOSTPARMS can have duplicate object IDs, as they are extensions of a data row under the nodes table with the same object ID.) Data rows with the same object ID can be duplicated across topology files in cases where a topology resides within another topology. The child and parent topologies have individual entries. (As mentioned previously, topology files have their object IDs as part of their filenames.)

A data row can also be rejected

```
/* ASCTOSQL.CMD */
/* (c) copyright IBM Corp., 1993      */
/* Licensed Materials - Property of IBM */
/* All Rights Reserved                 */

/* Get the source directory name and target database name. */
parse upper arg dpath dbname rest

if ((dpath = '?') | (dpath = '')) then
do
  say 'ASCTOSQL.CMD - Converts LAN NetView Start ASCII database'
  say 'files to data rows in a Database Manager database.'
  say 'An example call:'
  say ' '
  say 'ASCTOSQL directory-with-ASCII-files sql-database'
  say 'ASCTOSQL E:\NET1\ NETWORK'
  say ' '
  say 'The SQL database must have been created by '
  say 'IBM LAN NetView Start before this program is invoked.'
  exit
end

if \ (substr(dpath,length(dpath),length(dpath)) = '\') then
  dpath = dpath || '\'

/* Load REXX utilities and initialize variables */
call RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
call SysLoadFuncs
numtables = 15
tblname.1 = 'NETWORKS'
tblname.2 = 'TOPOLOGIES'
tblname.3 = 'FOLDERS'
tblname.4 = 'NODES'
tblname.5 = 'CONNECTIONS'
tblname.6 = 'ADAPTERPARMS'
tblname.7 = 'EMDFTPARGS'
tblname.8 = 'EMNONDFTPARGS'
tblname.9 = 'INCLUDES'
tblname.10 = 'HOSTPARMS'
tblname.11 = 'APPLICATIONS'
tblname.12 = 'RTRANSFORMERS'
tblname.13 = 'LCUTTRANSFORMERS'
tblname.14 = 'NVTRANSFORMERS'
tblname.15 = 'LCUMAP'
numsets=6
setname.1 = 'NODE_NAME_SET'
setname.2 = 'LOCAL_NODE_NAME_SET'
setname.3 = 'NODE_LAN_ADDRESS_SET'
setname.4 = 'DBWORKSTATION_NAME_SET'
setname.5 = 'LSCOMPUTER_NAME_SET'
setname.6 = 'NODE_DOMAIN_NAME_SET'

/* Delete any DEL and MSG files that already exist. */
do i = 1 to numtables
  len = 8
  if length(tblname.i) < 8 then
```

Figure 6. ASCTOSQL.CMD REXX command file (continued on page 121)



```
len = length(tblname.i)
delfilename.i = substr(tblname.i,1,len)
'erase ' delfilename.i || '.DEL '
'erase ' delfilename.i || '.MSG '
end

/* Find the topology files. */
fullpath = dpath || '*.TOP'
call SysFileTree fullpath, 'file', 'F'
do i = 1 to file.0
    filename.i = substr(file.i,38)
end
numfiles = file.0

/* There's only one network file to find. */
fullpath = dpath || '*.NET'
call SysFileTree fullpath, 'file', 'F'
numfiles = numfiles+1
filename.numfiles = substr(file.1,38)

/* Go through each of the files and store them
in the appropriate DEL file */
do i = 1 to numfiles
    say 'Processing File: ' || filename.i
    outfilename = 'NONE'
    do until lines(filename.i) = 0
        inline = linein(filename.i)
        skip = false
        do j = 1 to numtables
            if (tblname.j = substr(inline, 2, length(inline))) then
                do
                    outfilename = delfilename.j || '.DEL '
                    say '    On table: ' || tblname.j
                    skip = true
                end
            end
        end
    end

    /* if a set name was encountered, skip it and its data rows */
    do j = 1 to numsets
        if (setname.j = substr(inline, 2, length(inline))) then
            do
                outfilename = 'NONE'
                skip = true
            end
        end
    end

    if (skip = false & outfilename <> 'NONE') then
        do
            /* if in the topology section and the
            topology id isn't 2 or 3, skip the row */
            if ((outfilename = 'TOPOLOGI.DEL') |
                (outfilename = 'FOLDERS.DEL')) then
                do
                    topid = substr(inline, 1, (pos(',', inline) -1))
                    tempstr = translate(filename.i, ' ', '\')
                    tempstr = translate(tempstr, ' ', '.')
                    fileid = word(tempstr, (words(tempstr)-1))
                end
            end
        end
    end
end
```

Figure 6. ASCTOSQL.CMD REXX command file (continued on page 122)

due to the existence of double quotes within double quotes. This is valid for Start, since a user may include double quotes within a comment string; the program pairs the double quote with another double quote and delimits the entire string with double quotes. However, Database Manager does not allow a delimiting character to reside within a delimited string. Recovery from this error is easy; simply edit the data row to remove the double quotes within the string and retry the import.

The ASCTOSQL.CMD program can be enhanced to include additional error checks such as querying to see if a specified database already exists. An enhanced version of this program, ASCTOSQ2.CMD, is included as an applet with Start 1.1.

TRANSFERRING A NETWORK FROM SQL TO ASCII

Users can move from a DBM database to ASCII files using similar techniques. The user will need to perform the reverse of the preceding steps, but in this case there will be multiple topology files. Which data row needs to go in which file? The answer can be found by looking at the values in the TOPOBJECTID column, the third data value in data rows for all tables except the NETWORK table. There will be as many files as there are topologies. A separate network file must also be created.

First, put each of the network, topology, and folder data rows into a separate file, using the ASCII file name conventions (NET1.NET, 2.TOP, and so on) and the same file format (table name, data row, and table name). As the network file will contain only one table, the name sets must be included as placeholders for now. Next, since most topology and folder objects can have a parent (topologies within topologies), check if the topology data line has a TOPOBJECTID different from its OBJECTID. If so, duplicate the data row



under the appropriate table name in the parent file. The network name and address sets will need to be created in the network file by querying the database for valid non-template values. Finally, put each of the data rows in the remaining tables in the appropriate topology file under the correct table line.

SUMMARY

Transferring a network from ASCII files to a DBM database or vice versa is difficult and time-consuming if performed by hand, but with a REXX program using the DBM command-line interface to query rows and store them in files, these steps can be automated to make switching between the two databases easy. Users can have the speed of the Start ASCII database and the query and reporting facilities of the DBM database with the same network.

ACKNOWLEDGMENTS

We would like to give a big thanks to our fellow LAN NetView Start teammates: John Bunce, Mike Burkey, Phil Clem, Kristen Clarke, Eva Dea, George Dever, Rodger Fields, Mike Garrison, Mike Giboney, Ken Knight, Gena Linville, Rick Nicholas, Harold Nowak, Juan Nuncio, Oscar Perez, Glenn Schaefer, Chuck Schoepflin, Diane Skeel, Cheryl Slavin, Anthony Stevens, and Gale Wilson.

Theodore Shrader, IBM Personal Systems Line of Business, 11400 Burnet Rd., Austin, Texas, 78758. Shrader is a senior associate programmer; he joined IBM in June 1989, working on graphical interfaces to the Database Manager product. He has worked in graphical application development since then, most recently on the IBM LAN NetView Start program. He holds a B.S. in computer science from the University of Texas at El Paso. (Viva Miners!)

```

        if ((\topid = fileid) | (fileid = 2) | (fileid = 3)) then
            rc = lineout(outfilename, inline)
        end
    else
        rc = lineout(outfilename, inline)
    end
end

end
end /* do */

/* Close all the DEL files. */
do i = 1 to numtables
    rc = lineout(delfilename.i || '.DEL')
end

call dbm "STARTDBM"
call dbm "START USING DATABASE " || dbname
/* Import the DEL files into the database */
do i = 1 to numtables
    say 'Importing into table: ' || tblname.i
    call dbm "DELETE FROM " || tblname.i
    call dbm "IMPORT TO " || dbname || " FROM " || delfilename.i || ,
        ".DEL OF DEL REPLACE INTO " || tblname.i || " MESSAGES " || ,
        delfilename.i || ".MSG"
end
call dbm "STOP USING DATABASE"
/* call dbm "STOPDBM" */

say ' '
say 'ASCII to SQL DBM conversion complete.'
say 'Examine the *.MSG message files for import warning messages,'
say ' if any. A data row might be rejected because it contained'
say ' double quotes in a string value.'
```

Figure 6. ASCTOSQL.CMD REXX command file (continued from page 121)

Khalil Emami, IBM Personal Systems Line of Business, 11400 Burnet Rd., Austin, Texas, 78758. Emami is a senior associate programmer who joined IBM in November 1987, working on the Communications Manager product. He has worked in distributed computing environment (RPC) development since then and has most recently joined the IBM LAN NetView Start test and beta support group. He holds a B.S. in computer engineering and a B.A. in computer science from the University of Texas at Austin. He also has an M.S. in computer science from Southwest Texas State University.

REFERENCES

IBM LAN NetView Start User's Guide (IBM Doc. S96F-8585)

Automated Installation for CID-Enabled Extended Services, LAN Server v.3.0, and Network Transport Services/2 (IBM Doc. GG24-3781)

Automated Installation of CID-Enabled Products Using NetView DM/2 R4 and NetView DM/2 2.0



The advanced package of the LAN Server 3.0 includes a disk fault tolerance subsystem that protects a server from data loss due to fixed disk failures. Because fault tolerance support is integrated into the operating system, existing applications need no programming changes. **BY PAT TITTIZER**

Server-Based Systems: Protection from Disk Failures

FOR MANY COMPUTER SYSTEMS, THE FIXED disk is the most vulnerable component. A damaged fixed disk may result in the complete loss of data. Backups, even when done on a regular basis, may not contain all the lost data. Additionally, recovery from a failure may take a long time.

Protection from disk failures at a server is even more critical. A server, used by multiple people, may have several fixed disks. The disks often contain an organization's vital data that must always be available during working hours.

Over the years, many methods have been developed to prevent data loss from a damaged fixed disk. Many of the methods involve keeping extra copies of the data online through redundant arrays of inexpensive disks (RAID). Most RAID methods require additional purchases of application software or specialized (and expensive) hardware.

The advanced package of the IBM OS/2 LAN Server 3.0 includes a disk fault tolerance subsystem that implements RAID 1. This code works hand-in-hand with the 386 high-performance file system (HPFS) subsystem and the OS/2 2.0 DASD Manager™ to protect a server from data loss due to fixed disk failures. Because fault tolerance support is integrated into the operating system, existing applications need no programming changes. Any fixed disks supported by IBM OS/2 may be used with the fault tolerance subsystem.

Fault tolerance support includes:

- Drive mirroring and duplexing
- Automatic load balancing of read requests for improved performance
- Logging of disk-related errors
- Automatic correction of some errors
- Alerting network administrators when serious errors occur
- Local and remote configuration
- Local and remote administration.

*Any fixed disks supported by OS/2
may be used with the
fault tolerance subsystem.*

FAULT TOLERANCE FEATURES

Drive mirroring and duplexing provide duplication of data stored on a disk. Figure 1 shows an example of a server with fault tolerance configured, where drive F: is a mirrored drive consisting of a primary partition on the second physical disk and a secondary partition on the first physical disk. When data is written to drive F:, it is written to both partitions, automatically providing a duplicate copy. When data is read from drive F:, the system accesses it on the least busy disk. If a disk failure occurs, the data

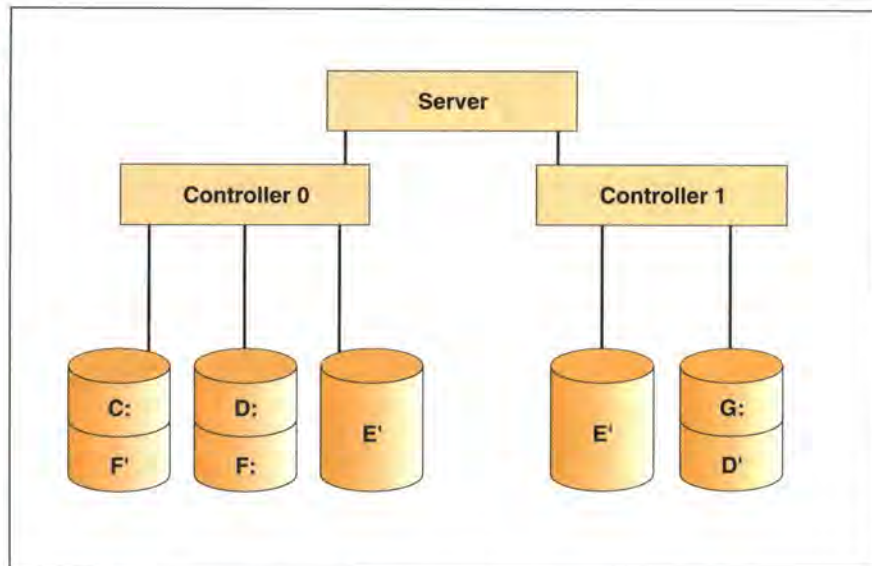


Figure 1. Example of a server with fault tolerance configured

can be recovered from the nonfailing partition.

Drive duplexing provides additional protection by using separate controllers. For example, in Figure 1, drive D: and drive E: are duplexed drives. Like mirrored drives, they consist of a primary partition on one disk and a secondary partition on another. The two disks are connected to different controllers, and if a controller failure occurs, read and write requests for these drives are routed to the partition connected to the good controller.

When a disk failure is detected, an error is logged. Serious errors will generate a network alert, if alerting has been configured. Some errors are automatically corrected. Others require administrator intervention for correction. When a complete drive failure is detected, the failing partition is disabled. Read and write requests are automatically routed to the good partition of the mirrored drive.

FAULT TOLERANCE SUBSYSTEM STRUCTURE

The complete fault tolerance subsystem structure is shown in Figure 5. The fault tolerance sub-

system consists of the following programs:

DISKFT.SYS. This fault-tolerant driver works with the OS/2 2.0 DASD manager to coordinate reads and writes to mirrored drives. At system startup, it determines the disk configuration, matching primary and secondary partitions. When disk errors occur, DISKFT initiates logging of the error. If necessary, DISKFT shuts down failed partitions. It also collects statistics about events on each drive.

FTMONIT.EXE. This fault monitor runs as a background process. When started, it checks file system structures for consistency on mirrored drives. When inconsistencies are detected, it logs an error and attempts to take corrective action. After the initial checks are done, FTMONIT goes to sleep, waiting for disk failures to be reported by DISKFT.SYS. When a failure is reported, FTMONIT logs an error and sends an alert.

FTSETUP.EXE. This Presentation Manager (PM)-based configuration program provides an interface for an administrator to configure

the server's drives. It provides mirroring, unmirroring, and deleting operations. Information about the location of each partition can also be displayed. Figure 2 shows an example of the FTSETUP main window.

FTADMIN.EXE. This PM-based administration program provides an interface for an administrator to correct disk failures, monitor disk statistics, and resynchronize mirrored drives. FTADMIN may be run either at the fault-tolerant server or remotely from a requester. Figure 3 shows an example of the FTADMIN main window.

FTREMOTE.EXE. This response file-driven configuration and administration program provides many of the same functions as FTSETUP and FTADMIN. FTREMOTE can be invoked at the local server or remotely via a software distribution utility such as the LAN CID. FTREMOTE is controlled by a response file that contains commands to configure drives and to correct errors. An example of a response file is shown in Figure 4. This response file unmirrors C:, mirrors D:, recovers the detached partition (a secondary partition without a corresponding primary partition) on disk 2 cylinder 1, and corrects all errors.

FT.DLL. This support function contains several service routines used by the other programs. It coordinates the information presented and used by the other fault tolerance programs.

Successful Read Requests. When a read request is made by 386 HPFS, the DASD Manager informs DISKFT. DISKFT determines which drive has been requested. If both sides of the mirror are healthy, DISKFT tells the DASD manager to read from either side of the mir-



ror. The DASD manager looks at the queue length for both fixed disks and schedules the read request for the least busy disk. In many systems, this will result in faster performance.

When the read request completes, the DASD manager informs DISKFT, which increments the drive statistics, determines the correct return code for the read, and tells the DASD manager that the read request is complete.

Successful Write Requests. When a write request is made by HPFS, the DASD manager informs DISKFT, which directs it to write to the primary partition. It also generates a second write request for the secondary partition.

When the DASD manager informs DISKFT that both write requests have been completed, DISKFT increments the drive statistics, determines the correct return code for the write, and tells the DASD manager that the write request is complete.

Read Failures. If the first read request completes unsuccessfully, the DASD manager informs DISKFT, which increments the drive statistics and tells the manager to read from the other side of the mirror. If the second read is completed successfully, DISKFT sets the return code to "read with recovery." If the second read is unsuccessful, DISKFT sets a failure return code.

When 386 HPFS receives the return code, it determines if a hot fix is needed. Hot fixing is done by marking the failing disk sector as error prone. The data from that sector is written to another location on the disk, and subsequent I/O is automatically directed to the new location.

DISKFT also tracks the number of failures. If 16 out of 32 requests fail, DISKFT informs FTMONIT of the problem and stops directing

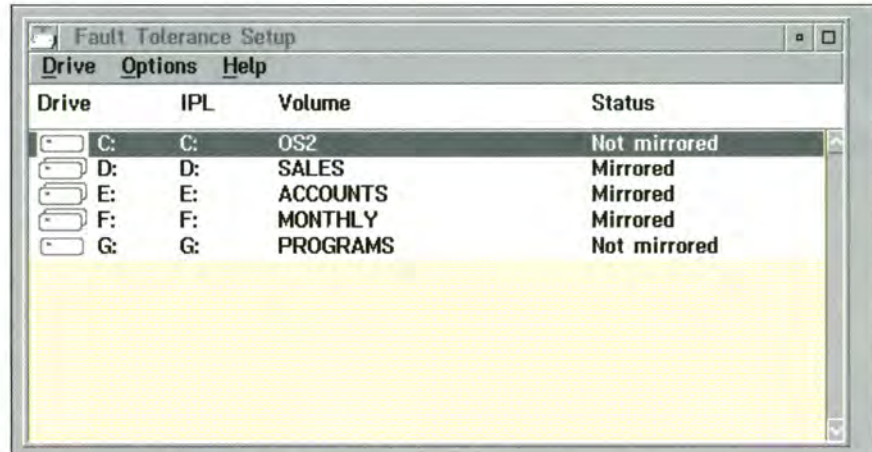


Figure 2. FTSETUP main window

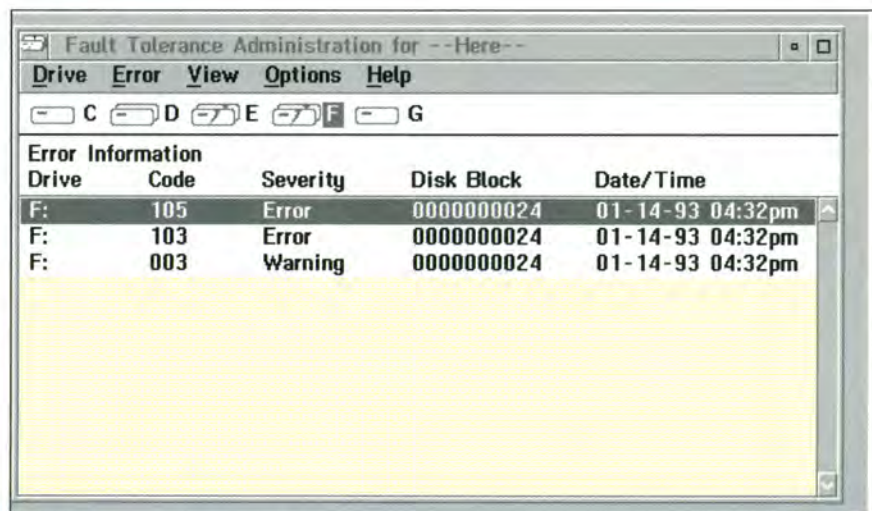


Figure 3. FTADMIN main window

read requests to the error-prone partition. FTMONIT logs an error and sends an alert.

Write Failures. If one of the two write requests completes unsuccessfully, DISKFT sets the return code to "write with recovery." If both write requests are unsuccessful, DISKFT sets a failure return code. As for read failures, 386 HPFS determines if a hot fix is needed.

Disk Failures. If errors continue to happen on an error-prone partition or if the DASD Manager is unable to access the partition, DISKFT shuts down the failing partition. All read and write requests are directed only to the

good side of the mirror. DISKFT informs FTMONIT of the shutdown drive, and FTMONIT logs an error and sends an alert.

Recovering From A Disk Failure. Recovering from a complete disk failure is not difficult, if all the drives contained on the failed disk were mirrored. An outline of the steps to recover from a disk failure are:

- Replace the failing disk.
- If the disk containing OS/2 2.0 has failed and if the partition is mirrored, use FTREMOTE from a 386 HPFS boot disk to recover the partition. Install the OS/2 Boot Manager and make the


```

<inline coding example>
CURRENT
  < Drive   Location  Status
    C      1:2(2:20)  Mirrored
    D      1:80      Not Mirrored
    ?      2:1        Detached  >
COMMANDS
  < RECOVER LOC=2:1 PRIMARY
    UNMIRROR C DELETE
    MIRROR D
    CORRECT ALL  >

```

Figure 4. Response file

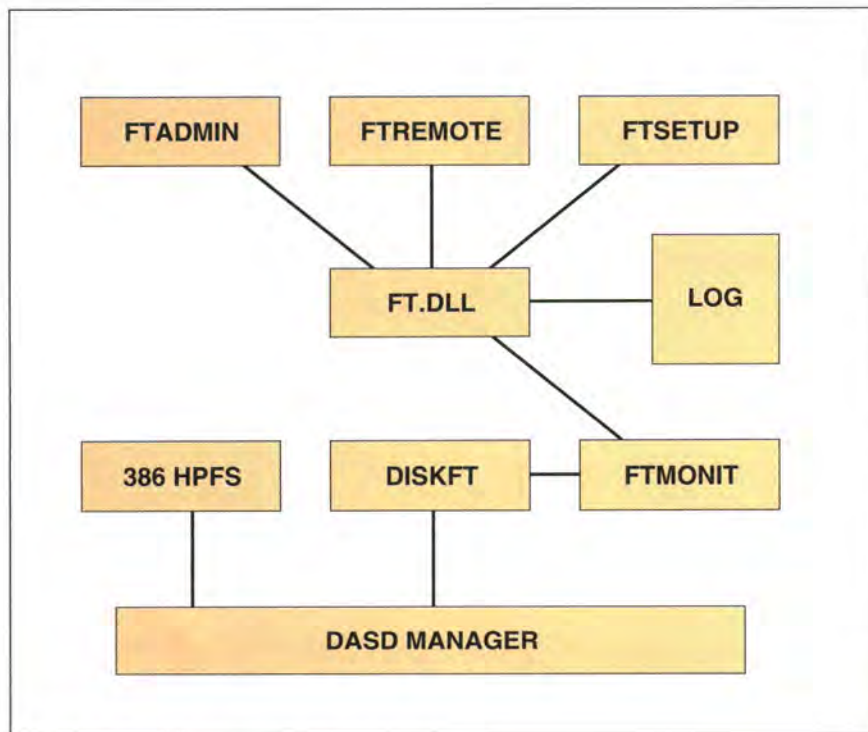


Figure 5. Fault tolerance subsystem structure

- partition startable.
- If the disk containing OS/2 has failed and the partition is *not* mirrored, reinstall OS/2 and the LAN Server.
- Run FTSETUP to recover all detached drives and to remirror drives that have lost their secondary partitions.
- Restart the computer. All your data and programs should now be available.

WHEN TO USE FAULT TOLERANCE

Disk fault tolerance should be

used on any server containing critical data. With drive mirroring, the system automatically keeps two up-to-date versions of the data. Failures of read operations are automatically corrected by reading your data from the other partition.

Even on systems with fault tolerance installed, a regular schedule of data backup should be maintained. While fault tolerance protects your data from single failures, it may not provide protection for multiple failures or ones that occur on both partitions of the mir-

rored drive.

With some RAID systems, the boot drive cannot be protected. This leaves the boot drive vulnerable. LAN Server's fault tolerance support is capable of mirroring the boot drive. Therefore, it should be used on RAID systems that have unprotected boot drives.

Disk fault tolerance should be used on any server that requires high availability. If a fixed disk should fail, the fault tolerance code will automatically configure itself to use only the good partitions. In most cases, the computer can keep operating until the failed disk can be replaced.

Duplexing provides additional protection, allowing the computer to keep operating when a disk controller fails.

Pat Tittizer, IBM Personal Systems Products Division, 11400 Burnet Rd., Austin, Texas, 78758. Tittizer is an advisory programmer in the OS/2 LAN Server team and has been with IBM for 20 years. She has a B.S. in mathematics from San Jose State University and an M.S. in computer sciences from the University of Texas, Austin.

REFERENCES

Alford, Roger C. "Disk Arrays Explained," *Byte Magazine*, October 1992, 259-266.

OS/2 LAN Server 3.0 Network Administrator Reference Volume 1: Planning and Installation (IBM Doc. S96F-8428).

OS/2 LAN Server 3.0 Network Administrator Reference Volume 3: Network Administrator Tasks (IBM Doc. S96F-8430).

Winship, Sally. "Disk-Mirroring Products Offer True Fault Tolerance," *PC Week*, February 4, 1991, 112-117.

The design of a thread-reentrant memory allocation scheme introduces two little known but highly useful kernel functions, `DosSubAllocMem` and `DosSubFreeMem`. Sample code fragments illustrate the concepts described.

BY LARRY SALOMON JR.

Writing Memory Allocation Functions With `DosSubAllocMem`

WHEN MICROSOFT AND IBM RELEASED the first OS/2-enabled C compilers, application developers were forced to reconsider their development strategies when writing multithreaded programs because many C run-time library functions were nonreentrant. Critical sections or serialization via semaphores were required to access these functions. Additionally, many developers were daunted by the need for different include files and libraries and the difficulty of developing multithreaded dynamic link libraries.

While some of today's OS/2 compilers, notably IBM's C Set/2 and Microsoft C 6.0, include thread-reentrant run-time libraries that make it unnecessary to work around this design limitation, replacing functions provided by the run-time library can teach programmers about kernel functions they might not use otherwise. In this article, the design of a thread-reentrant memory allocation scheme introduces two little known but highly useful kernel functions, `DosSubAllocMem` and `DosSubFreeMem`. Sample code fragments will illustrate the concepts described.

DOING YOUR HOMEWORK

Experienced developers know that implementation is but a single step on the path of application development. While each programmer generally has his or her own checklist of items to be done during this process, I have listed my guidelines here:

1. Describe the function to be provided.
2. Define the necessary data structures to provide the required functionality and do any internal housekeeping.
3. Write the programming interface (function prototypes and data structure names).
4. Develop a strategy for the implementation phase using pseudocode.
5. Implement the functions.

Some Notes

This article assumes that the reader is familiar with C language and has some programming experience with the OS/2 kernel APIs. The source code was developed under OS/2 2.0 and compiled as a 32-bit library with IBM's C Set/2. However, since there is a direct correlation between the 32-bit calls used in the library and their 16-bit counterparts, the complete source code supports conditional compilation to either 16- or 32-bit routines.

The complete source code described in this article can be obtained under the file name SALOMON.TXT on the Internet via anonymous ftp at the ftp-os2.nmsu.edu site in the /pub/os2/incoming directory, on CompuServe in the OS2DF2 forum library, and on the IBM National Support Center BBS, (404) 835-6600.

6. Refine the functions by using them in actual programs (versus predefined test cases) and distribute preliminary copies of the libraries and application to other developers or users.
7. Document the system.

First there was the Pony Express...



NOW... Announcing
ANIMATED ICONS for
Icon **EXPRESS** OS/2 2.0



Icon **EXPRESS** Utility for use with OS/2 2.0:

Fast... Changes icons on the desktop with a click of the mouse. Instead of 12 steps to change an icon, **Icon EXPRESS** reduces the number of steps to 1! Simply choose the icon to be changed, drag it to the new icon and click!

Organize... To organize your icons, simply create a new page in the **Icon EXPRESS** notebook and bring icons from other applications into these pages.

Create... See your icons come alive by creating your own Animated Icons using the Icon Express Editor.



Includes *Super* 1000 Icon Library:

Animations	Documents	Folders
Computer Devices	Sports Figures	Nature
Business/Office	Geometric Designs	Tools
	Alphabet Letters	



To Order Call:
1-800-ACT-7185

American Computer Technologies
2301 Maitland Center Parkway,
Suite 445
Maitland, Florida 32751

(919) 233-0716.....	43	(918) 357-2869.....	23
Borland International Inc.		MSR Development	
(408) 469-0115.....	COV 4	(409) 560-5964.....	55
Creative Systems Programming Corp.		One-Up Corp.	
(609) 234-1920.....	32	(214) 620-9626.....	25
Essex Systems Inc.		Real Time Technologies	
(508) 750-4699.....	71	(708) 446-6886.....	71
Gpf Systems Inc.		SE International Inc.	
(203) 873-2171.....	45	(407) 997-8945.....	19
Hilbert Computing		Sigma Imaging Systems Inc.	
(913) 829-2450.....	71	Softbridge Inc.	
Hippo Software		(617) 864-7747.....	35
(801) 531-1302.....	32	Stac Electronics	
IBM-Workshops.....	9	(619) 431-1001.....	COV 3
IBM-CUA Controls Library		Soft and GUI Inc.	
(919) 469-4423.....	128	(718) 934-2133.....	41
IBM- Canada.....	77	Softronics Inc.	
IBM- Personal Software Products		(719) 548-1878.....	28
(407) 982-6408.....	26	Sourceline Software Inc.	
IBM- TIRS.....	43	(619) 587-4713.....	37
Information Builders Inc.		Sundial Systems Corp.	
Impact Software		Symantec	
(818) 879-5593.....	71	(310) 453-0636.....	31
Intelligent Environments Inc.		The Stirling Group	
(508) 640-1090.....	47	(708) 307-9340.....	15
JDS Publishing		Token Technology Inc.	
(908) 247-0952.....	19	(415) 965-8658.....	33
Kaseworks		UCANDO Software Inc.	
(404) 448-4163.....	11	(919) 380-0757.....	21
Lookout Mountain Software		Watcom	
(719) 545-8572.....	71	(519) 747-4971.....	COV2-1
Magus		XVT Software Inc.	
(415) 940-1238.....	43	(303) 443-0969.....	16
Micro Focus			
(415) 856-6134.....	2		

Moving up to OS/2 from Windows? Let CUA Controls Library/2 deliver your GUI baggage.

If you're moving up to more powerful OS/2® from Windows™, now you can have the same user interfaces for both new and existing applications.

With CUA™ Controls Library/2 (CCL/2).

It's a 16-bit program that quickly generates familiar OS/2 2.0 GUIs for applications in OS/2 1.3 and Windows 3.0 and 3.1.

It provides consistent interfaces across platforms, so your end users will quickly learn new applications.

CCL/2 contains the code for seven key GUI screens. It's code you won't have to write. It's also code you can reuse. Over and over again.

What's more, CCL/2 will save you money. Because there are no runtime charges for redistribution of DLLs.

And because now it's available at the low price of only \$149. To order, call 1 800 342-6672.

For information, fax 1 919 469-4423.

So get moving. With CCL/2.



IBM and OS/2 are registered trademarks and CUA is a trademark of International Business Machines Corporation. Windows is a trademark of Microsoft Corporation. © 1993 IBM Corp.


```
(USHORT)DosSubAllocMem(PVOID pvBase,
                        PPVOID ppvPtr,
                        ULONG u1SzBlock);

(USHORT)DosSubFreeMem(PVOID pvBase,
                      PVOID pvPtr,
                      ULONG u1SzBlock);
```

Figure 1. DosSubAllocMem() and DosSubFreeMem() prototypes

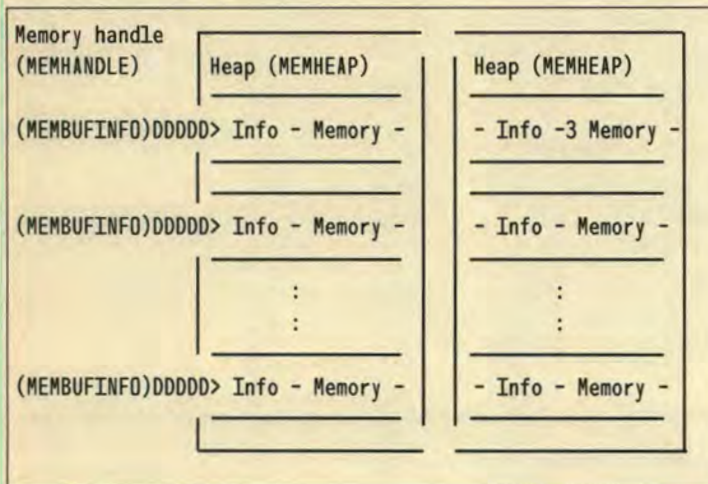


Figure 2. Graphical representation of the data structures

Step six I consider the most important because it reveals the true nature of a system—the well-roundedness of the functions, the robustness of the code, and the extent of functional limitations.

The importance of step three, however, should not be underestimated. It has always been said that first impressions are important, and this is true even in program design. Following SAA-like conventions for function names helps (the names are more intuitive) and costs nothing. For example, although `malloc` seem no less readable than `MemAlloc`, consider `cvtfb` versus `ConvertFoobar`. (This procedure should not be taken to extremes, however; it was once said that if SAA conventions were strictly followed, a function to open a pipe to connect two processes would be called `DosOpenPipeToConnectTwo Processes`.)

LAYING THE GROUNDWORK

Before we can begin coding, we must consider the design of our internal data structures, which are directly affected by the functions we wish to provide. For simplicity's sake, let us define the minimum functionality as:

- Allow the user to allocate a memory block of a specified size
- Allow the user to free a previously allocated memory block
- Allow the user to query the size of a previously allocated memory block.

In addition, it would be nice to take advantage of OS/2's ability to allocate blocks of memory of up to 512MB in size. Finally, a reset function to free all currently allocated memory can come in handy.

Knowing the System. The design of

any function is constrained by the actions allowed by the operating system, so we should stop and look at the `DosSubAllocMem` and `DosSubFreeMem` APIs, shown in Figure 1, since we will ultimately be using them to perform the memory allocation.

`DosSubAllocMem` is used to subdivide a block of memory previously allocated with `DosAllocMem`. Significantly faster than `DosAllocMem`, `DosSubAllocMem` manipulates an existing block of memory instead of creating a new one. It has a drawback, however, in that it provides no boundary checking; accessing the 101st byte of a 100-byte block allocated with `DosSubAllocMem` will not cause an error, but will likely corrupt all the sub-blocks in the memory pool.

`DosSubFreeMem` is used to free a block of memory previously allocated with `DosSubAllocMem`. Since the size of the block to be freed is required for the call, the developer must store this information when the memory is initially allocated so the user doesn't need to. Additionally, there should be a way to determine to which memory pool each block belongs.

It has always been said that first impressions are important, and this is true even in program design.

Press Enter to Continue. The size of each memory block should be noted somewhere. It is best to allocate more space than was requested—*m* bytes for the application and *n* bytes for housekeeping information, placed at the beginning of

the memory block.

Because `DosSubAllocMem` requires a memory block previously allocated with `DosAllocMem`, another data structure is needed to contain this “memory heap” (also referred to as a “memory pool”). If this structure contains a usage count for each heap, the pool can be returned to the system when it is no longer needed.

Finally, because avoiding global variables allows maintenance of thread reentrancy, encapsulate instance data in a private data structure. The user will need to provide a pointer (or handle) to this private data on each call to our routines. Figure 2 gives a graphical representation of various data structures, while Figure 3 shows the definitions of these structures.

LET'S BEGIN

Figure 4 shows the function prototypes of our memory routines. All of the functions return a value of type `MEMERROR`, consistent with the OS/2 kernel's philosophy of returning the result of a function call. The exception to this rule is the `MemQueryMemSize` routine, which returns the size of a previously allocated block of memory. (It could indeed return a `MEMERROR`, but since it isn't possible to determine if its argument was allocated with `MemAllocMem`, it would always return the same `MEMERROR` value.)

It is often easier to start at the ends and work your way inward; we will begin by examining the logic behind `MemInitialize` and `MemTerminate` that respectively create and destroy a `MEMHANDLE`.

MemInitialize and MemTerminate.

The memory handle points to a private data structure used by all the memory routines. `MemInitialize` simply needs to allocate an area of memory to hold this data structure, then initialize the contents to an empty state. The definition of the

```
typedef struct _MEMHANDLEINFO {
    USHORT usMaxHeaps;          /* Maximum number of heaps */
    ULONG uLSzHeap;             /* Size of each heap */
} MEMHANDLEINFO, *PMEMHANDLEINFO;

typedef struct _MEMBUFINFO {
    ULONG uLRequested;          /* Size requested */
} MEMBUFINFO, *PMEMBUFINFO;

typedef struct _MEMHEAP {
    PVOID pvHeap;               /* Points to memory to be used */
    ULONG uLUsage;              /* Counts number of accesses */
} MEMHEAP, *PMEMHEAP;

#define SIG_MEMHANDLE           (ULONG)0x12340000

typedef struct _MEMHANDLE {
    ULONG uLSignature;          /* Signature */
    MEMHANDLEINFO mhiInfo;      /* User-specified parameters */
    MEMHEAP amhHeaps[1];        /* Array of heaps */
} MEMHANDLE, *HMEMHANDLE;
```

Figure 3. Data structure definitions

```
typedef USHORT MEMERROR;

#define MEM_EC_NOERROR          (MEMERROR)0
#define MEM_EC_ERROR           (MEMERROR)1
#define MEM_EC_BADHANDLE       (MEMERROR)2
#define MEM_EC_NOMEMORY        (MEMERROR)3
#define MEM_EC_SIZETOOLARGE    (MEMERROR)4
#define MEM_EC_BADPOINTER      (MEMERROR)5

MEMERROR MemInitialize(PMEMHANDLEINFO pmhiInfo,PHMEMHANDLE phmMem)
MEMERROR MemTerminate(HMEMHANDLE hmMem);
MEMERROR MemReset(HMEMHANDLE hmMem);
MEMERROR MemAllocMem(HMEMHANDLE hmMem,ULONG uLSzBuf,PVOID *ppvBuf);
MEMERROR MemFreeMem(HMEMHANDLE hmMem,PVOID pvBuf);
ULONG MemQueryMemSize(PVOID pvBuf);

#define MEM_QH_ERROR            (USHORT)0
#define MEM_QH_HMEMHANDLE       (USHORT)1

MEMERROR MemQueryHandle(PVOID pvHandle);
```

Figure 4. Function prototypes

`MEMHANDLE` structure in Figure 3 specifies an array of `MEMHEAP`s of size one (`amhHeaps`), while we will likely need more than this. A `MEMHANDLE` is used

instead of a variable-length structure, and `amhHeaps` is defined simply to allow easy access to each of the `MEMHEAP` structures.


```

static PVOID myMemAlloc(ULONG ulSzBuf)
//-----
// This function calls DosAllocMem to allocate a block of memory.
//
// Input:  ulSzBuf - specifies the size of the block to allocate
// Returns: a pointer to the memory if successful, NULL otherwise
//-----
{
    PVOID pvBuf;

    if (DosAllocMem(&pvBuf,ulSzBuf,PAG_READ PAG_WRITE PAG_COMMIT OBJ_TILE
)) {
        return NULL;
    } /* endif */

    memset(pvBuf,0,ulSzBuf);
    return pvBuf;
}

static BOOL myMemFree(PVOID pvBuf)
//-----
// This function calls DosFreeMem to free a block of memory.
//
// Input:  pvBuf - points to the memory to free
//-----
{
    if (pvBuf==NULL) {
        return FALSE;
    } else
    if (DosFreeMem(pvBuf)) {
        return FALSE;
    } else {
        return TRUE;
    } /* endif */
}

MEMERROR MemInitialize(PMEMHANDLEINFO pmhiInfo,PHMEMHANDLE phmMem)
//-----
// This function creates an instance of memory subsystem.
//
// Input:  pmhiInfo - points to a MEMHANDLEINFO structure describing the
//           desired attributes of the instance to be created.
// If
//           NULL, the defaults are used.
//           phmMem - points to the variable that is to receive the result
// Output:  phmMem - points to the variable containing the memory handle
// Returns: MEM_EC_NOERROR if successful, an error code otherwise
//-----
{
    MEMHANDLEINFO mhiInfo;
    ULONG ulSzBuf;

```

Figure 5. MemInitialize() and MemTerminate() (continued on page 132)

MemTerminate is even simpler to understand than MemInitialize. It destroys all the heaps currently being used via a call to MemReset and frees the memory being used by its argument. (myMemAlloc and myMemFree are referenced in the other routines, which is why we made them separate functions instead of calling DosAllocMem and DosFreeMem directly.)

MemQueryHandle returns a value describing the type of its argument. It examines the first four bytes pointed to by its argument, which are assumed to specify a unique signature for each handle type. This trick is unnecessary here since there is only one type of handle in this subsystem; however, it can come in handy in other applications with multiple handle types.

MemReset frees the memory used by all the heaps, examining the values of the pvHeap field of each MEMHEAP structure and calling myMemFree if pvHeap is not NULL. Although this function is infrequently used, it can be very useful. As an example, consider a drawing application that allocates a block of memory for each object in a drawing, such as circles, squares, and so on. If the user starts a new drawing, it would help to call a single function to delete all objects in the last drawing viewed.

MemAllocMem and MemFreeMem.

The two functions that form the meat of the subsystem are more complex and involved than the other functions we've described, but they aren't much more difficult to write. The pseudocode for these functions is shown in Figure 6, and their source code in Figure 7. In MemFreeMem, notice the reference to the macro HIUSHORT. In myMemAlloc, we allocated the memory of each heap with the attribute OBJ_TILE that tells OS/2 to allocate the memory on a 64k boundary, essentially guaranteeing that the upper half of the memory pointer is unique with-

in our process. This makes it easy to search the heaps for the one containing the memory buffer to be freed (as long as the size of the memory block being referenced does not exceed 64K).

Pay attention to the `ulUsage` field in the `MEMHEAP` structure. Whenever this reference count, which is maintained by `MemAllocMem` and `MemFreeMem`, reaches zero in `MemFreeMem`, we call `myMemFree` to return the memory to the system.

WRAPPING IT UP

We mentioned that the problem with the original implementation of the C run-time library was that most of the routines were not thread-reentrant. While the code here is mostly thread-reentrant, there is still the possibility of a data access collision. (For example, one thread could call a function that modifies a data structure, while another thread calls a function that reads the same structure.) This problem could be solved by including a mutex semaphore handle in the `MEMHANDLE` structure, requesting the semaphore at the entrance of each function and then releasing the semaphore before returning.

The `MemReset`, `MemQueryMemSize`, and `MemQueryHandle` functions have not been implemented here. They are implemented in the full source (see the sidebar on page 127 for information on how to obtain it from several electronic bulletin board systems).

This article has stepped through the process of designing and implementing a memory allocation subsystem that can be easily incorporated into existing applications. It has described how the `DosSubAllocMem` and `DosSubFreeMem` functions work, their advantages and disadvantages, and one way to design an application so they can be used with little trouble. Finally, it has detailed some necessary considerations when writing a reusable function library.

```
*phmMem=NULL;

//-----
// If the caller specified NULL for pmhiInfo, set the defaults.
//-----
if (pmhiInfo!=NULL) {
    mhiInfo=*pmhiInfo;
} else {
    mhiInfo.usMaxHeaps=256;
    mhiInfo.ulSzHeap=61440;
} /* endif */

ulSzBuf=sizeof(MEMHANDLE)+sizeof(MEMHEAP)*(mhiInfo.usMaxHeaps-1);

*phmMem=myMemAlloc(ulSzBuf);
if (*phmMem==NULL) {
    return MEM_EC_NOMEMORY;
} /* endif */

(*phmMem)->ulSig=SIG_MEMHANDLE;
(*phmMem)->mhiInfo=mhiInfo;
memset((*phmMem)->amhHeaps,0,mhiInfo.usMaxHeaps*sizeof(MEMHEAP));
return MEM_EC_NOERROR;
}

MEMERROR MemTerminate(HMEMHANDLE hmMem)
//-----
-
// This function destroys an instance of memory subsystem.
//
// Input:  hmMem - specifies the handle of instance to destroy
// Returns: MEM_EC_NOERROR if successful, an error code otherwise
//-----
-
{
    if (MemQueryHandle(hmMem)!=MEM_QH_HMEM) {
        return MEM_EC_BADHANDLE;
    } /* endif */

    //-----
    // Destroy all of the heaps being used by this handle
    //-----
    if (MemReset(hmMem)!=MEM_EC_NOERROR) {
        return MEM_EC_ERROR;
    } /* endif */

    myMemFree(hmMem);
    return MEM_EC_NOERROR;
}
```

Figure 5. `MemInitialize()` and `MemTerminate()` (continued from page 131)

MemAllocMem:

Verify that the memory handle is valid and that the memory size requested isn't larger than the size of the heap (minus 64 bytes for OS/2's housekeeping and minus the size of the MEMBUFINFO structure for our own housekeeping)

Loop

Check each heap to see if we can satisfy the request. If so, allocate the memory, increment the heap's reference count and return

End

We couldn't satisfy the request with the existing heaps, so create a new heap if we haven't reached the maximum number of heaps

Allocate the memory, set the reference count for the heap to 1, and return

MemFreeMem:

Verify that the memory handle is valid

Search through the heaps to find the heap that contains the buffer to be freed

Free the memory and decrement the heap's reference count
If the heap's reference count has reached zero, destroy the heap
Return

Figure 6. MemAllocMem() and MemFreeMem() pseudocode

```
#define SZACTUAL(u1SzBuf)      (ULONG)((u1SzBuf+sizeof(MEMHEAP))+
                                8-((u1SzBuf+sizeof(MEMHEAP))%8)
)

MEMERROR MemAllocMem(HMEMHANDLE hmMem,ULONG u1SzBuf,PVOID *ppvBuf)
//-----
// This function attempts to allocate an amount of memory specified by
// the caller from a heap.
//
// Input:  hmMem - specifies the memory handle to allocate from
//         u1SzBuf - specifies the amount of memory requested
//         ppvBuf - points to the variable to receive the result
// Output: ppvBuf - points to the allocated memory
// Returns: MEM_EC_NOERROR if successful, an error code otherwise
//-----
{
    ULONG u1SzActual;
    USHORT usFirstNull;
    USHORT usIndex;
    PMEMBUFINFO pmbiInfo;
    PMEMHEAP pmhHeap;

    if (MemQueryHandle(hmMem)!=MEM_QH_HMEMHANDLE) {
```

Figure 7. MemAllocMem() and MemFreeMem() (continued on page 134)


```

    return MEMERROR_EC_BADHANDLE;
} /* endif */

*ppvBuf=NULL;

//-----
// Adjust the buffer size to account for the MEMHEAP structure and for
// the rounding that OS/2 will do anyways. Check to see if the result
// is larger than the size of the heap-64 bytes for OS/2's housekeeping
//-----
ulSzActual=SZACTUAL(ulSzBuf);
if (ulSzActual>hmMem->mhiInfo.ulSzHeap-64) {
    return MEMERROR_EC_SIZETOOLARGE;
} /* endif */

//-----
// usFirstNull is the index of the first unused heap. Initialize it to
// indicate that we haven't found one yet.
//-----

usFirstNull=hmMem->mhiInfo.usMaxHeaps;

for (usIndex=0; usIndex<hmMem->mhiInfo.usMaxHeaps; usIndex++) {
    if (hmMem->amhHeaps[usIndex].pvHeap!=NULL) {

        //-----
        // Since this heap isn't NULL, attempt to satisfy the request.
        // If it succeeds, increment the reference count for the heap,
        // initialize our housekeeping info, set ppvBuf to point to the
        // memory just beyond our housekeeping info, and return
        //-----

        if (!DosSubAllocMem(hmMem->amhHeaps[usIndex].pvHeap,
                           (PVOID)&pmbiInfo,
                           ulSzActual)) {
            hmMem->amhHeaps[usIndex].ulRef++;

            pmbiInfo->ulSzRequested=ulSzBuf;

            *ppvBuf=(PVOID)(pmbiInfo+1);
            return MEMERROR_EC_NOERROR;
        } /* endif */
    } else
    if (usFirstNull==hmMem->mhiInfo.usMaxHeaps) {

        //-----
        // We found an unused heap, so set usFirstNull to usIndex if we
        // haven't found one previously.
        //-----
        usFirstNull=usIndex;
    } /* endif */
} /* endfor */

```

Figure 7. MemAllocMem() and MemFreeMem() (continued on page 135)


```

//-----
// If we make it here, then none of the heaps could satisfy the request.
// If we didn't find an unused heap, then return a "no memory" error.
//-----
if (usFirstNull==hmMem->mhiInfo.usMaxHeaps) {
    return MEMERROR_EC_NOMEMORY;
} /* endif */

pmhHeap=&hmMem->amhHeaps usFirstNull;
pmhHeap->ulRef=0;

//-----
// Allocate a new heap, initialize it for suballocation (via
// DosSubSetMem), and allocate the memory from the new heap.
//-----

pmhHeap->pvHeap=myMemAlloc(hmMem->mhiInfo.ulSzHeap);
if (pmhHeap->pvHeap==NULL) {
    return MEMERROR_EC_NOMEMORY;
} /* endif */

if (DosSubSetMem(pmhHeap->pvHeap,
                DOSSUB_INIT,
                hmMem->mhiInfo.ulSzHeap)) {
    myMemFree(pmhHeap->pvHeap);
    return MEMERROR_EC_NOMEMORY;
} /* endif */

if (!DosSubAllocMem(pmhHeap->pvHeap,
                    (PVOID)&pmbiInfo,
                    ulSzActual)) {
    pmhHeap->ulRef++;

    pmbiInfo->ulSzRequested=ulSzBuf;

    *ppvBuf=(PVOID)(pmbiInfo+1);
    return MEMERROR_EC_NOERROR;
} else {
    return MEMERROR_EC_NOMEMORY;
} /* endif */
}

MEMERROR MemFreeMem(HMEMHANDLE hmMem,PVOID pvBuf)

//-----
// This function frees memory previously allocated by MemAllocMem
//
// Input:  hmMem - specifies the memory handle to free from
//         pvBuf - points to the memory to free
// Returns: MEM_EC_NOERROR if successful, an error code otherwise
//-----
{
    USHORT usIndex;
    PMEMBUFINFO pmbiInfo;

```

Figure 7. MemAllocMem() and MemFreeMem() (continued on page 136)


```

if (MemQueryHandle(hmMem)!=MEM_QH_HMEMHANDLE) {
    return MEMERROR_EC_BADHANDLE;
} /* endif */

//-----
// Search the heaps for the one containing the memory buffer to free
//-----

for (usIndex=0; usIndex<hmMem->mhiInfo.usMaxHeaps; usIndex++) {
    if (HIUSHORT(hmMem->amhHeaps usIndex|.pvHeap)==HIUSHORT(pvBuf)) {

        //-----
        // We found the heap containing this memory buffer, so free the
        // memory and decrement the reference count
        //-----

        pmbiInfo=((PTBHEAPINFO)pvBuf)-1;

        if (DosSubFreeMem(hmMem->amhHeaps usIndex|.pvHeap,
                        pmbiInfo,
                        SZACTUAL(pmbiInfo->ulSzRequested))) {
            return MEMERROR_EC_ERROR;
        } /* endif */

        hmMem->amhHeaps usIndex|.ulRef--;

        if (hmMem->amhHeaps usIndex|.ulRef==0) {
            myMemFree(hmMem->amhHeaps usIndex|.pvHeap);
            hmMem->amhHeaps usIndex|.pvHeap=NULL;
        } /* endif */

        return MEMERROR_EC_NOERROR;
    } /* endif */
} /* endfor */

return MEMERROR_EC_BADPOINTER;
}

```

Figure 7. MemAllocMem() and MemFreeMem() (continued from page 135)

Larry Salomon, 89-17 Moline St., Bellerose, N.Y. 11428. Salomon wrote his first Presentation Manager application for OS/2 in 1989. Since that time, he has writ-

ten several VIO and PM applications, including the Scramble applet included with OS/2 and the I-Brow/Magnify/Screen Capture trio, included with the IBM Profes-

sional Developer's Kit CD-ROM. Currently, he works for International Masters Publishers in Stamford, Conn. He can be reached in Internet at os2man@panix.com.

SO, YOU HAVE OS/2... NOW, DO YOU HAVE THE ROOM TO USE IT?

AVAILABLE
APRIL, 1993



Introducing Stacker for OS/2 & DOS

If you're running OS/2®, you've probably discovered how much disk space your powerful new operating system demands.

No worry!

With Stacker® for OS/2 & DOS, you can quickly and safely double the capacity of your hard disk to take full advantage of OS/2's power.

Using Stac® Electronics' patented, award-winning Stacker LZS™ technology,

Stacker instantly and transparently compresses all your data, "stacking" it more efficiently so you have more room to work.

That's all there is to it! You can install Stacker in minutes and access all your

stacked files at any time, whether you boot from OS/2 or DOS.

And, with the Stacker Optimizer™, you can quickly defragment your stacked drives to get the best possible performance. Stacker works on disks as big as one gigabyte, giving you up to two gigabytes of disk capacity. It even comes with a simple Unstack command that returns your system to its original, unstacked state.

New Stacker for OS/2 & DOS. Think of it as elbow room for your operating system.

CALL NOW FOR MORE INFORMATION
1-800-522-STAC, ext. 8204
or 619-431-7474, ext. 8204
Fax 619-431-9616

STAC



Undefeated champion!

Borland C++ leaves pretenders playing catch-up.

REPORT CARD		INFO
AUGUST 17, 1992		WORLD
C++ Windows development		
 Borland C++ & Application Frameworks version 3.1		 Microsoft C/C++ version 7.0
Performance		
Programming environment	Excellent	Good
Language	Very good	Good
Productivity	Excellent	Good
Documentation	Very good	Excellent
Ease of learning	Very good	Good
Ease of use	Excellent	Good
Support		
Support policies	Excellent	Very good
Technical support	Very good	Very good
Value	Excellent	Very good
Final scores	8.9	6.9

Borland C++ consistently outscores Microsoft C/C++ in performance, ease of use, and value.

6362-0001-17

Borland C++ & Application Frameworks™ is unquestionably the best C and C++ compiler and tools for building applications. That's why it is the clear winner in every review and product comparison. And why more than one million professional programmers rely on Borland C++.

Why is Borland C++ the best? Because its highly visual tools and Integrated Development Environment make it easy to create high-performance Windows applications. And since Borland C++ is the only popular compiler that meets the certified* ANSI C and AT&T C++ standards, you know your investment in Borland C++ will take you well into the future.

Now in its third generation, Borland C++ gives you the features and reliability you can trust for DOS, Windows, and now OS/2.*



Go with Borland C++, the #1-selling C and C++ application development system.

See your dealer or call now, 1-800-331-0877, ext. 5042

In Canada, call 1-800-461-3327.



Borland® C++
for Windows, DOS, and OS/2

*ANSI C certification awarded by the British Standards Institute. Copyright © 1993 Borland International, Inc. All rights reserved. All Borland product names are trademarks of Borland International, Inc. B1 4787.1

